National Library of Canada
Canadian Theses Service
Ottawa, Canada
K1A 0N4

Bibliothèque nationale du Canada
Service des thèses canadiennes

NOTICE

The quality of this microform is heavily dependent upon the quality of the original thesis submitted for microfilming. Every effort has been made to ensure the highest quality of reproduction possible.

If pages are missing, contact the university which granted the degree.

Some pages may have indistinct print especially if the original pages were typed with a poor typewriter ribbon or if the university sent us an inferior photocopy.

Previously copyrighted materials (journal articles, published tests, etc.) are not filmed.

Reproduction in full or in part of this microform is governed by the Canadian Copyright Act, R.S.C. 1970, c. C-30.

AVIS

La qualité de cette microforme dépend grandement de la qualité de la thèse soumise au microfilmage. Nous avons tout fait pour assurer une qualité supérieure de reproduction.

S'il manque des pages, veuillez communiquer avec l'université qui a conféré le grade.

La qualité d'impression de certaines pages peut laisser à désirer, surtout si les pages originales ont été dactylographiées à l'aide d'un ruban usé ou si l'université nous a fait parvenir une photocopie de qualité inférieure.

Les documents qui font déjà l'objet d'un droit d'auteur (articles de revue, tests publiés, etc.) ne sont pas microfilmés.

La reproduction, même partielle, de cette microforme est soumise à la Loi canadienne sur le droit d'auteur, SRC 1970, c. C-30.

NL-339 (r. 88/04)

Canada

THE UNIVERSITY OF ALBERTA

APPLICATIONS OF LEARNING HIERARCHIES IN ADAPTIVE INSTRUCTIONAL SYSTEMS

by

© JOHN CALE NESBIT

A THESIS

SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE
OF DOCTOR OF PHILOSOPHY

DEPARTMENT OF EDUCATIONAL PSYCHOLOGY

EDMONTON, ALBERTA
SPRING, 1988

Permission has been granted to the National Library of Canada to microfilm this thesis and to lend or sell copies of the film.

The author (copyright owner) has reserved other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without his/her written permission.

L'autorisation a été accordée à la Bibliothèque nationale du Canada de microfilmer cette thèse et de prêter ou de vendre des exemplaires du film.

L'auteur (titulaire du droit d'auteur) se réserve les autres droits de publication; ni la thèse ni de longs extraits de celle-ci ne doivent être imprimés ou autrement reproduits sans son autorisation écrite.

ISBN 0-315-42837-6

University of Alberta
Edmonton
Canada T6G 2G5

Division of Educational Research Services
Faculty of Education
3-104 Education Building North, Telephone (403) 432-3762

March 4, 1988

We, J. C. Nesbit and S. Hunka, authors of the article "A method for sequencing instructional objectives which minimizes memory load" published in volume 16 of *Instructional Science*, hereby give permission to J. C. Nesbit to include a modified version of the article in his thesis "Applications of learning hierarchies in adaptive instructional systems".

John Nesbit
J. C. Nesbit

S. Hunka
S. Hunka

THE UNIVERSITY OF ALBERTA

RELEASE FORM

NAME OF AUTHOR: John Cale Nesbit

TITLE OF THESIS: Applications of Learning Hierarchies in Adaptive Instructional Systems

DEGREE: Doctor of Philosophy

YEAR THIS DEGREE GRANTED: Spring, 1988

Permission is hereby granted to THE UNIVERSITY OF ALBERTA LIBRARY to reproduce single copies of this thesis and to lend or sell such copies for private, scholarly or scientific research purposes only.

The author reserves other publication rights, and neither the thesis nor extensive extracts from it may be printed or otherwise reproduced without the author's written permission.

John Nesbit

4655 Keith Road
West Vancouver, B.C.
Canada V7W 2M8

Date: March 2, 1988

THE UNIVERSITY OF ALBERTA

FACULTY OF GRADUATE STUDIES AND RESEARCH

The undersigned certify that they have read, and recommend to the Faculty of Graduate Studies and Research, for acceptance, a thesis entitled Applications of Learning Hierarchies in Adaptive Instructional Systems submitted by John Cale Nesbit in partial fulfillment of the requirements for the degree of Doctor of Philosophy.

S. Hunka

Supervisor

Marlene L. Jones

Michael Szabo

EW Romaniuk

Date:
February 27th, 1988

To my parents

Abstract

Learning hierarchies, which are often used by instructional designers to represent the prerequisite structure of course objectives, are a practical way of organizing adaptive instructional systems which deal with global course structure. In this thesis, two new methods are developed for use in such systems. One of these builds on the sequencing function of learning hierarchies, and the other on the diagnostic function.

A learning hierarchy often permits a large number of alternate linear arrangements (sequences) of instructional objectives. The premise of the sequencing method developed here is that for every sequence a memory load value can be calculated which is theoretically related to the probability that students will fail to recall prerequisite objectives. An algorithm is presented which generates a minimal memory load sequence from a learning tree, a restricted but frequently encountered type of learning hierarchy. To assess the effectiveness of the algorithm in generating low memory load sequences given hierarchies which are not trees, it was applied to several published examples of learning hierarchies. The results indicated that the algorithm is effective as a heuristic, especially when combined with a hill-descending procedure which attempts to incrementally improve the generated sequence.

When a student exhibits difficulty in learning an objective, the learning hierarchy serves a diagnostic function by indicating prerequisites, one or more of which may have been forgotten or never mastered. An inductive learning procedure was developed which, after encountering several students, will associate a profile or model of the student with the state (mastered or not mastered) of the prerequisites. An evaluation of the performance of the procedure with artificial student models showed that, under certain conditions, the classification rules it learns will reduce the amount of testing required to determine which prerequisites are not mastered. Guidelines for using the procedure and areas in need of improvement are also indicated.

ACKNOWLEDGEMENTS

Thanks are firstly due to my supervisor Dr. S. Hunka, whose substantial contribution to the work reported here is but a small fraction of efforts he has extended in support of countless graduate students over the years. I owe special gratitude to Dr. E. W. Romaniuk who, in addition to serving on the supervisory committee, first introduced me to the field of computer assisted instruction through his introductory course and through an assistantship with the Division of Educational Research Services. For their insightful suggestions, I would like to thank the committee members: Drs. A. Liu, R. Elio, D. Baine, M. Szabo, and M. Jones. I would also like to acknowledge constructive comments by fellow graduate student H. Chow on an early version of Chapter 3.

TABLE OF CONTENTS

Chapter Page

1 Introduction 1
  1.1 Learning hierarchies as a representation of global course structure 2
  1.2 An overview of this thesis 4
  1.3 References 5

2 Adaptive Instructional Systems 6
  2.1 Attribute-treatment interaction and beyond 8
  2.2 Hartley's framework for adaptive instructional systems 11
  2.3 Learner control 12
  2.4 The cost of program controlled adaption 13
  2.5 Smallwood's teaching machine 15
  2.6 Atkinson's drill optimization system 23
  2.7 Adaptive computer-managed instruction 26
  2.8 Tennyson's MAIS 28
  2.9 Ferguson's branch and test procedure 31
  2.10 Knowledge-based adaptive sequencing in BIP 33
  2.11 Koffman and Perry's concept selection procedure 34
  2.12 Heines and O'Shea's rule-based tutor 36
  2.13 Peachey and McCalla's plan-based CAI 39
  2.14 Conclusion 42
  2.15 References 43

3 A Method for Sequencing Instructional Objectives which Minimizes Memory Load 47
  3.1 Learning hierarchies as acyclic digraphs 48
    3.1.1 Level numbers 51
    3.1.2 Inreach and outreach 51
    3.1.3 Interpretation and treatment of inessential arcs 51
    3.1.4 Augmented learning hierarchies 52
  3.2 Memory load 53
  3.3 Deficiencies of memory load as a model of forgetting 55
  3.4 A sequencing algorithm which minimizes memory load 56
  3.5 Generalizing the algorithm to deal with non-trees 58
  3.6 Conclusion 62
  3.7 References 63

4 Inductive Learning and Adaptive Instruction 65
  4.1 Inductive learning 66
    4.1.1 Inductive learning by parameter adjustment 68
    4.1.2 Inductive inference and description spaces 69
    4.1.3 Learning conjunctive concepts 72
    4.1.4 Learning disjunctive concepts 77
    4.1.5 Learning from noisy examples 88
  4.2 An inductive learning procedure for adaptive instructional systems 92
    4.2.1 Classification rules and recommendation criteria 96
    4.2.2 Attributes 97
    4.2.3 The description language 98
    4.2.4 The inductive learning procedure 98
  4.3 References 105
5 Performance of the Inductive Learning Procedure 108
  5.1 A close-up view of ILP at work 109
  5.2 Four simulated cases for testing ILP 113
  5.3 Pre-generalization 120
  5.4 The effect of varying search parameters 121
  5.5 The effect of noise 122
  5.6 Initialization of descriptions 125
  5.7 Other observations 126
  5.8 Guidelines for the use of ILP 126
  5.9 Future research 127
  5.10 References 128
6 Application and Further Development of Memory Load Sequencing, ILP, and Related Procedures 129
  6.1 Application and development of planning procedures 130
  6.2 Application and development of diagnostic procedures 133
  6.3 References 133
Appendix A 134
Appendix B 135

LIST OF TABLES

Table Page

1.1 Periods in CAI (adapted from Goldstein and Carr, 1977) 2

3.1 The GENERATOR algorithm 57

3.2 Performance of GENERATOR 60

4.1 The candidate elimination algorithm learning a simple concept 76

4.2 The positive example deletion strategy 79

5.1 A comparison of number of incorrect recommendations with pre-generalization (pg) and without pre-generalization (npg) 121

5.2 Effect of increasing the search on number of incorrect recommendations 121

LIST OF FIGURES

Figure Page
2.1 An attribute - treatment interaction....10
2.2 A general branching network....16
2.3 A decision tree for a teaching machine....20
3.1 A simple digraph....49
3.2 A learning hierarchy....50
3.3 The arc from A to C is inessential....52
3.4 A hierarchy with nodes labeled by GENERATOR....59
4.1 Part of a description space partially ordered by the more-general-than relation....72
4.2 A beam search with maximum beam width = 2....74
4.3 The boundary sets S and G are used to compactly define the version space....75
4.4 Generalization hierarchies determined by specifying attribute type....82
4.5 A CLS/ID3 decision tree....86
5.1 Modules and files developed for performance evaluation....108
5.2 Descriptions learned after 9 examples....110
5.3 Descriptions learned after 10 examples....110
5.4 Descriptions learned after 56 examples....111
5.5 Descriptions learned after 100 examples....112
5.6 A case where a single example produces a radical change in the description....113
5.7 Performance of ILP on Case 1....115
5.8 Performance of ILP on Case 2....117
5.9 Performance of ILP on Case 3....119
5.10 Performance of ILP on Case 4....119
5.11 Varying the search parameters with Case 4....122
5.12 Performance of ILP on Case 1 with 0%, 5%, and 10% noise....123
5.13 Performance of ILP on Case 2 with 0%, 5%, 10% noise....124
5.14 Performance of ILP on Case 3 with 0%, 5%, and 10% noise....124
5.15 Performance of ILP on Case 4 with 0%, 5%, and 10% noise....125
6.1 Transforming a conjunctive learning hierarchy into a conjunctive module hierarchy....132

Chapter 1

Introduction

In this thesis two new methods are developed with the purpose being to improve the sequencing and diagnostic functions of learning hierarchies in adaptive instructional systems. Throughout the thesis, "adaptive instructional system" (AIS) is used rather than the more current "intelligent tutoring system" (ITS) because the former term suggests continuity with work prior to 1970 which is often ignored in recent literature. Another distinction adopted here is that, while ITS conventionally refers to software designed to teach specific content, an AIS has a somewhat broader scope.

The need to program domain-specific tutoring techniques is indeed the greatest deficiency of current ITS. So far, the application of artificial intelligence (AI) to education seems to have required that, in order to exploit domain-specific knowledge, different techniques be developed for every new instructional program. In practical terms, little has changed since Goldstein and Carr (1977) presented the tongue-in-cheek sketch of "periods of CAI" reproduced in Table 1.1

The main reason why it has been so difficult to break away from the ad hoc nature of domain-specific ITS is that instructional theory, on which any such advance must depend, is very weak. Good teachers acquire most of their art, not from studying a science of learning, but through experience teaching specific subjects. The tutoring heuristics they learn do not necessarily generalize well to new subjects, or even to new lessons in the same subject. The approach advocated here is that we provide the instructional system with the little practical knowledge that instructional theory has to offer, and also endow it with means to acquire domain-specific knowledge as it teaches. However, in the foreseeable future, we cannot hope to entirely eliminate the need for course authors and the domain-specific software they produce.

Table 1.1 Periods in CAI (adapted from Goldstein and Carr, 1977).

Primitive	Classical	Romantic	Modern
Programmed instruction (using workbooks)	PLATO TICCIT	SCHOLAR SOPHIE set theory with EXCHECK WUMPUS BIP	
no computer	no domain-specific expertise	use of AI for expertise	AI applied to theory of learning and teaching

1.1 Learning hierarchies as a representation of global course structure

In order to use general instructional principles, there must be a knowledge representation structure applicable to many varieties of subject matter. The learning hierarchy structure developed by Gagne (1962) is a promising candidate for part of such a representation. Learning hierarchies have gained wide acceptance among instructional designers. If frequency of presentation in instructional design textbooks is a valid measure, the learning hierarchy is second in importance only to behavioral objectives as a hallmark of systematic instructional design (e.g., De Cecco, 1968; Dick & Carey, 1978; Briggs & Wager, 1981; Martin & Briggs, 1986).

In a review of learning hierarchy research, White and Gagne (1974) stated that "studies since 1961 have provided fairly consistent support for Gagne's hierarchy postulate, and as the methodology of investigations has been improved and the limits of the postulate have been clarified, this support has strengthened"[1]. This interpretation of the research was confirmed by a meta-analysis (Horon & Lynn, 1980) of 15 studies which tested the effectiveness of learning hierarchies in the sequencing of objectives. Horon and Lynn concluded that the use of learning hierarchies substantially increases achievement and marginally reduces learning time.

---

[1] By "limits of the postulate" they mean Gagne's original contention that learning hierarchies are applicable only to intellectual skills, and not to verbal knowledge.

Modified forms of the Gagne learning hierarchy have figured prominently in a few AIS (Westcourt, Beard & Gould, 1977; Heines & O'Shea, 1985; Peachey & McCalla, 1986). Like the work reported in this thesis, these systems are based on the concept that learning hierarchies can serve as a good framework for building large courses (i.e., exceeding 50 student hours) from collections of related modules, where each module covers a specified set of objectives. In the scheme adopted here, a module may be any instructional presentation that a course author[2] deems appropriate and feasible, from a single frame to an entire ITS. In some circumstances, the nesting of hierarchies in modules would also be useful.

Learning hierarchies are a practical way of representing global course structure. They involve little extra labour for the author because most courses covering intellectual skills are, or should be, subjected to a hierarchical analysis in the design phase. This is an important advantage because lengthy authoring time is already the single greatest barrier to the proliferation of good courseware.

Also, this approach allows for an integration of current instructional design practices with a variety of instructional methods. The only constraint imposed on such methods is that they provide an assessment of student mastery of the objectives they are designed to teach. Accommodation of diverse methods is important because no single paradigm currently dominates the field of computer-based education. The last 25 years has seen several promising and stimulating applications of artificial intelligence to educational problems but, as Suppes (1984) has observed, intelligent CAI (ICAI) is still in its infancy and is characterized by inflated expectations and a certain naive disregard for stochastic methods. There is no comparative research to support claims concerning the superiority of ICAI over frame-based CAI, but there is a great deal of research supporting the hypothesis that conventional CAI is as effective or more effective than standard classroom instruction (Bangert-Drowns, Kulik & Kulik, 1985; Niemiec, Samson, Weinstein & Walberg, 1987; Niemiec & Walberg, 1987). The point is that, given the current state of computer-based education, a system dealing with global course structure is more likely to be successful if it places few limitations on instructional methods used in local modules.

---

[2] Throughout the thesis, the term "author" refers to anyone involved in the design, implementation, or maintenance of a course.

1.2 An overview of this thesis

The purpose of the work reported here was to develop new ways of exploiting learning hierarchies in adaptive instructional systems. It is hoped that these procedures will eventually be available from course authoring and execution facilities.

Chapter 2 is a review of previous work on adaptive instructional systems. It focuses on methods which are relatively independent of subject area, and especially on those which use learning hierarchies.

Chapter 3 proposes an extension of the standard method for sequencing instructional objectives with learning hierarchies. The extension tries to minimize a simple psychological construct called memory load. Procedures are developed which are practical for large learning hierarchies.

Chapter 4 presents an inductive learning procedure (ILP) aimed at improving the diagnostic function of learning hierarchies. When certain assumptions are met, ILP will substantially reduce the amount of testing required to identify which objectives from a set of prerequisites are in a state of non-mastery.

An experimental evaluation of ILP using simulated subjects is reported in Chapter 5. The evaluation indicated guidelines for usage of ILP and areas in need of further work.

In Chapter 6, consideration is given to the integration of memory load sequencing and ILP into future AIS. Further development of these and other procedures is suggested.

1.3 References

Bangert-Drowns, R., Kulik, J., & Kulik, C. (1985). Effectiveness of computer-based education in secondary schools. *Journal of Computer-Based Instruction*, 12(3), 59-68.

Briggs, L., & Wager, W. (1981). *Handbook of Procedures for the Design of Instruction*. Englewood Cliffs NJ: Educational Technology.

De Cecco, J. (1968). *The Psychology of Learning and Instruction: Educational Psychology*. Englewood Cliffs NJ: Prentice-Hall.

Dick, W., & Carey, L. (1978). *The Systematic Design of Instruction*. Glenview IL: Scott, Foresman.

Gagne, R. (1962). The acquisition of knowledge. *Psychological Review*, 69(4), 355-365.

Heines, J., & O'Shea, T. (1985). The design of a rule-based CAI tutorial. *International Journal of Man-Machine Studies*, 23, 1-25.

Horon, P., & Lynn, D. (1980). Learning hierarchies research. *Evaluation in Education*, 4, 82-83.

Martin, B., & Briggs, L. (1986). *The affective and cognitive domains*. Englewood Cliffs NJ: Educational Technology.

Niemiec, R., Samson, G., Weinstein, T., & Walberg, H. (1987). The effects of computer based instruction in elementary schools: A quantitative synthesis. *Journal of Research on Computing in Education*, 20(2), 85-103.

Niemiec, R., & Walberg, H. (1987). Comparative effects of computer-assisted instruction: A synthesis of reviews. *Journal of Educational Computing Research*, 3(1), 19-37.

Peachey, D., & McCalla, G. (1986). Using planning techniques in intelligent tutoring systems. *International Journal of Man-Machine Studies*, 24, 77-98.

Suppes, P. (1984). Observations about the application of artificial intelligence research to education. In D. Walker and R. Hess (Eds.). *Instructional Software: Principles and perspectives for design and use*. Belmont CA: Wadsworth.

Westcourt, K., Beard, M., & Gould, L. (1977). Knowledge-based adaptive curriculum sequencing for CAI: Application of a network representation. *Proceedings of the Association for Computing Machinery*, Seattle WA, p. 234-239.

White, R., & Gagne, R. (1974). Past and future research on learning hierarchies. *Educational Psychologist*, 11(1). 19-28.

Chapter 2

Adaptive Instructional Systems[3]

There is no consensus on definition of the term **adaptive instruction**, but a rough characterization of the concept can be conveyed by a few of the many overlapping definitions given by theorists:

> Adaptive instructional strategies facilitate individual learning by adjusting the basic learning environment to the unique learning characteristics and needs of each learner. . . . For an instructional system to be adaptive, it must be both diagnostic and prescriptive.
>
> (Rothen & Tennyson, 1978, p. 317)

> Adaptive materials characteristically: (a) save the students' time and effort by letting them skip unneeded teaching material; (b) test students to determine their needs; and (c) reflect individual differences among the students.
>
> (Holland, 1977, p. 147)

> The term "adaptive" is used to imply that the student is routed to material which follows different teaching strategies and that tasks are put together, on-line, to fit a particular student's competence. The components of an adaptive instructional system are, representations of tasks and student performances, and a set of teaching operations which are controlled by means-ends guidance rules.
>
> (Hartley, 1973, p. 421)

Two features separate the modern study of adaptive instructional systems (AIS) from the wider field of individualized or adaptive education: (1) an AIS is ultimately expressed in the form of a computer program (even though, as in the case of computer-managed instruction, some other medium may deliver the instruction) and (2) an AIS represents an attempt to individualize the way in which the curriculum is presented, but not the curriculum itself. By many definitions, virtually any CAI program with canned text and response contingent branching can be classified as an AIS. But in this chapter, a definition is adopted which agrees with Gable and Page's (1980) historical classification of teaching programs: an AIS accomplishes individualization through some systematic and widely applicable technique which exploits a student model formed from a history of responses which is stored between instructional sessions.

---

[3] A version of this chapter appeared as a Division of Educational Research Services Research and Information Report (RIR-87-3).

A crude taxonomy of adaptive instructional systems is suggested by several, admittedly fuzzy, parameters which researchers have allowed to vary in attempting to individualize instruction:

- Pace. This refers to the rate at which instructional messages are presented to the learner and may be measured as words per minute, pages per hour, and so on. Self-pacing was often cited as an important advantage of programmed instruction (PI), teaching machines, and CAI in the early 1960's. Except in cases where skills such as reading speed are being trained (Wilkinson, 1983), the universally preferred approach to individualizing the pace of instruction is to place it under learner control.
- Amount of instruction. This is the number of examples or degree of detail in a presentation required by a student to learn an objective. It is inversely related to step size. The term step size arose in the context of PI where it referred to "the amount of increase in subject matter difficulty with each step in the program" (Cook, 1961, p. 153).
- Amount of practice. This parameter is similar to amount of instruction, but it can be adjusted according to measures of fluency derived from practice responses.
- Selection of instructional objectives to be covered. The primary goal of many AIS developers has been to target the instruction to the knowledge state of the learner. In practice this means skipping instruction on concepts which the AIS has reason to assume the student knows.
- Sequence. Once needed objectives have been identified, some order for presenting them (or tasks which incorporate them) must be determined. For example, BIP II (Westcourt, Beard, & Gould, 1977) used a learning hierarchy, an estimate of the student's learning ability based on previous performance, and a list of objectives

from the hierarchy not yet mastered, to select the best BASIC programming task to present next.

- Learning style. This is really a catch-all factor which covers many kinds of individual differences. For example, in Heines & O'Shea's tutorial, learning style was interpreted as student preferences for non-interactive demonstration, assigned exercises, or discovery learning.

Most adaptive instructional systems start from the same state with each new student they encounter, and there is no capacity for applying to later students information gleaned from dealing with the current student. A **self-improving** AIS (Hartley & O'Shea, 1973) tries to bring previous experience to bear on the problem of adapting to a new student. The mechanism for accomplishing this can range from the rote learning of response data (e.g., Smallwood, 1962), to rule learning by inductive inference as presented in Chapter 4 of this thesis.

This review will primarily cover adaptive instructional systems which have been designed to deal with a wide range of content areas. Tutoring programs which incorporate adaptive techniques are of interest here only to the extent that those techniques can be generally applied. Before taking a fairly detailed look at several content-independent adaptive instructional systems, a very brief historical overview is given of significant developments in educational psychology bearing on the issue of individualized education, and a few general issues relating to these systems are presented.

2.1 Attribute-treatment interaction and beyond

With the advent of universal compulsory education in Europe and North America in the late nineteenth and early twentieth centuries, educators were challenged on an unprecedented scale with the problem of impressing a standard curriculum on a large and diverse student population. In his influential paper "The child and the curriculum" which appeared in 1902, Dewey argued for the Rousseauan ideal that the child be the "the starting point, the center, and the end" of the educational process (Dewey, 1976, p. 276). Both Dewey's essay and Thorndike's book *Individuality*, first published in 1911, reflected the growing recognition of human diversity and of the commensurate need for individualization of instructional methods and curriculum (Glaser, 1977). This attitude was reinforced by the results of the administration of a modified version of Binet's new intelligence test to 1.75 million U.S. army recruits in World War I (Gould, 1981).

Over the following four decades, school systems introduced means of channeling secondary students into a small number of different programs (e.g., business, vocational, academic) according to their career expectations or tested abilities. In his 1957 presidential address to the American Psychological Association, Cronbach suggested that psychology had been prevented from helping to bring about a more profound change by a schism which had divided the discipline since the early 1920's. The split was along the ancient nativist-empiricist fault line: correlational psychology, which deals with variation in ability or aptitude, versus experimental psychology, which deals with variation in treatment.

Cronbach proposed a new agenda for applied psychology:

> The job of applied psychology is to improve decisions about people. The greatest social benefit will come from applied psychology if we can find for each individual the treatment to which he can most easily adapt. This calls for the joint application of experimental and correlational methods. . . . For any practical problem there is some best group of treatments to use and some best allocation of persons to treatments. We can expect some attributes of persons to have strong interactions with treatment variables. These attributes have far greater importance than attributes which have little or no interaction.
>
> (Cronbach, 1957, p. 679)

Figure 2.1 illustrates Cronbach's point about attribute treatment interaction. Treatment A produces greater average achievement than treatment B, but the greatest average achievement or payoff is attained when students are divided into two treatment groups according to aptitude. The point is that, when an ATI is present, it is possible to do better than to simply give everyone the single treatment which produces the greatest average achievement.

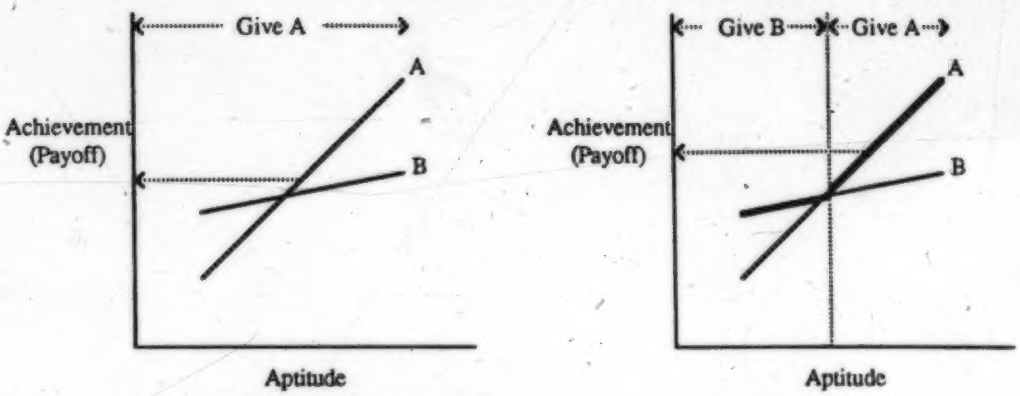


Figure 2.1 An attribute - treatment interaction.

However, the disordinal interactions necessary to realize the potential of this enlightened view had yet to be discovered. Although a search for ATIs was pursued intently by many researchers in the sixties and seventies, there were few interesting results (Glaser & Resnick, 1972). In surveying 90 ATI studies involving 108 potential ATIs, Bracht (1970) found that only 5 significant disordinal interactions had been reported. All but one of the ATIs found were not applicable to instruction. Cronbach and Snow (1977, p. 6) conceded that "well-substantiated findings regarding ATI are scarce", and suggested that the original concept of ATI, though an improvement on the single treatment approach, had been an oversimplification.

Several limitations in the original ATI model were noted by Cronbach and Snow (1977) and Tobias (1976):

- An ATI may involve more than one aptitude or treatment variable.
- The set of attributes interacting with a treatment variable may vary as the treatment progresses.
- The attribute measured in a pre-test may not remain stable over the period of time in which the treatment occurs.
- Evidence indicated that an ATI validated for one content area is often not generalizable to other topics.

In the face of these possibilities, a successful ATI hunt would seem to require guidance from stronger theories of instruction than are presently available.

Whatever Cronbach's original intention, attribute was often narrowly interpreted as a static heritable trait. Tobias (1976) observed that in ATI studies where prior familiarity with the content was included in the analysis, disordinal interactions were often found, not with the expected aptitude, but with the familiarity variable. He proposed that achievement-treatment interactions be used to determine treatment assignment. The shift from trait to state implies that treatments be short, allowing for frequent treatment assignments to keep up with the current state of the student.

2.2 Hartley's framework for adaptive instructional systems

The most enduring paradigm for the structure of an adaptive instructional system is due to Hartley (1972; 1973; Hartley & O'Shea, 1973). Hartley's framework is best viewed not as an AIS itself, nor as a prescription for AIS structure, but as a descriptive tool allowing for qualitative comparisons between systems and bringing a limited degree of order to a chaotic field.

Hartley's framework is based on the General Problem Solver (GPS), a pioneering AI program developed by Newell, Shaw, & Simon (1960). Given a current state, a desired state, domain-dependent operators, and domain-dependent heuristics for applying the operators, GPS tried to obtain through a "means-ends analysis" an operator sequence for transforming the current state to the desired state.

By analogy, a system for solving instructional problems requires the following components:

1) A task representation (or expert model)
2) A student model
3) A teacher model composed of
    - a set of teaching operations

- a set of means-ends guidance rules (heuristics for applying the operations)

The goal of the instructional system is to obtain an ordered set of teaching operations capable of transforming the student model into the expert model. This framework assumes the existence of a mechanism for updating the student model as instruction proceeds. The guidance rules are reinvoked when an updated version of the student model becomes available.

2.3 Learner control

There are three potential loci of control in an instructional system. Decisions can be made by a human teacher, a computer program (the teacher model), or the learner. The term **learner control** usually implies manipulation by the student of sequence, amount of instruction, or mode of instruction. Learner control is best exemplified by TICCIT, a CAI system tailored to the teaching of concepts (Merrill, Schneider & Fletcher, 1980). In TICCIT, students could choose which concept to study and whether to learn about the concept by rule, example, or practise. As the primary alternative to an AIS as a means of individualizing computer-based instruction, the prospects for learner control are now briefly considered.

The rationale behind learner control is that, through introspection, the student has access to internal states that can at best be only crudely modeled by an AIS. It is also claimed that giving control to learners makes them more motivated and leads to higher achievement. The counter-argument is that these advantages, if they exist at all, are defeated by shortcomings in the other two areas of necessary knowledge: knowledge of how to teach one's self (the teacher model), and knowledge of what is being taught (the expert model).

Experimental evidence bearing on these points is insufficient to arrive at any general conclusion. In a review of the sparse research evaluating the effects of learner control in CAI, Steinberg (1977) found that the achievement of learner control groups was lower or not significantly different from that of groups with less control. Some studies found learner control subjects to have more favorable attitudes toward CAI while showing poorer achievement.

The solution to the locus of control issue would seem to be a judicious mix of learner, teacher, and program control (Nakayama & Higashibara, 1986). Both Steinberg (1977) and Snow (1980) recommended that degree of learner control be itself regarded as a treatment variable to be individualized, and that attributes should be sought which interact with the locus of control variable. There is some weak evidence supporting the intuitive notion that learner control works best with high achieving, or better informed students (Tennyson, 1981). A precept of this dissertation, which is supported by Tennyson's study, is that the possibility of learner control does not negate the value of research on program control because even when the system does not control, it should be able to advise.

2.4 The cost of program controlled adaption

Some adaptive instructional programs (e.g., IPI Mathematics) try to minimize learning time by administering a diagnostic test on objectives not yet covered and presenting instruction only on objectives for which the student failed to demonstrate mastery. Holland (1977) pointed out that there is no advantage in employing such an adaptive decision when the instructional time it saves is largely offset by additional time spent in the test. All formative tests are candidates for this kind of criticism, because even a post-test can be seen as a kind of pre-test allowing those who pass to skip remedial instruction. As a quantification of this principle, Holland presented a consequence ratio:

$$\frac{I}{I+T}$$

where I is the estimated time to teach the objective and T is the estimated time to test the objective. A consequence ratio of 0.5 was taken to be the "break-even" point, with lower values representing adaptive decisions which cost more time than they save.

Holland calculated average consequence ratios of four individualized instructional programs including Atkinson's (1968) influential early reading CAI program. In Atkinson's program, every teaching frame is also a diagnostic test which branches the student either ahead to the next teaching frame, or to one of several corrective frames depending on his or her response. Holland obtained an average consequence ratio for this program of 0.70 which he concluded was "a disappointing consequence ratio for a program sparking a multi-million dollar CAI movement".

But Holland's analysis is seriously flawed. Aside from errors made in dealing with the non-dichotomous nature of Atkinson's CAI frames, there is a problem with the basic formulation of the consequence ratio. It accounts for the cost to students who pass the test, but ignores the cost incurred by those who fail it. A better measure of the utility of an adaptive decision is the expected time saving per student:

$$I - (T + qI)$$

where q is the proportion of students failing the test.

Unfortunately, the validity of this improved metric rests on the same questionable prior assumptions used by Holland. He argued that a single item cannot properly serve both diagnostic and instructional purposes because good instructional items must have very low failure rates, and good diagnostic items must have failure rates close to 50%. The premise that instructional items should be errorless is an axiom of the philosophy of PI (Skinner, 1958) to which Holland is a long time adherent. The current mood of educational psychology, though no better supported by empirical evidence on this point[4], backs the somewhat opposing view that uncertainty and conceptual conflict provide necessary motivation (Gagne, 1985). Although the best level of item difficulty for learning is debatable, few would claim that nothing can be learned from items of maximum discriminability. The importance of high discriminability for criterion-referenced diagnostic items is also questionable.

---

[4] Hartley (1973) developed a system capable of adjusting task difficulty to maintain predetermined success rates. Of three treatment groups with success rates of 60%, 75%, and 90%, the group maintained at 75% showed highest achievement, with the 90% group outperforming the 60% group.

Implicit in Holland's analysis is the belief that the only cost resulting from giving a student more instruction than he or she needs to achieve mastery is the time taken to do so. But there is evidence (Tennyson & Rothen, 1977) that excess instructional items can lead to lower post-test scores.

The validity of the diagnostic test should be included in any formulation of the utility of the adaptive decision. Cronbach and Gleser employed validity coefficients in the utility functions considered in their wide-ranging attempt (Cronbach & Gleser, 1957) to unify psychological testing with decision theory. It is likely that the instructional decision problem posed by Holland is a special case covered by their taxonomy of psychological decision problems.

2.5 Smallwood's teaching machine

Smallwood (1962) developed the earliest self-improving AIS. Although the design of his system was strongly influenced by contemporary research on teaching machines and programmed instruction, the ability of the system to use experience to improve its decision criteria distinguishes it from previous systems. An examination of the essential aspects of Smallwood's teaching machine is warranted by its influence on later instructional control systems.

In Smallwood's system, concepts to be taught are arranged in a linear sequence which is fixed for all students. For each concept, the course author prepares a **general branching network** and a post-test on the concept. A general branching network is comprised of **blocks**, which can be roughly described as frames of instruction: each having an instructional message and a question exercising and testing the student on the information presented in the message.

Smallwood assumes that in learning a single concept students progress through an invariant sequence of levels of understanding. It is also assumed that every block can be recognized as transferring students from a certain lower level to certain higher ones. Figure 2.2 shows an example of a general branching network. The jth block starting at the ith level is designated b_{ij}.

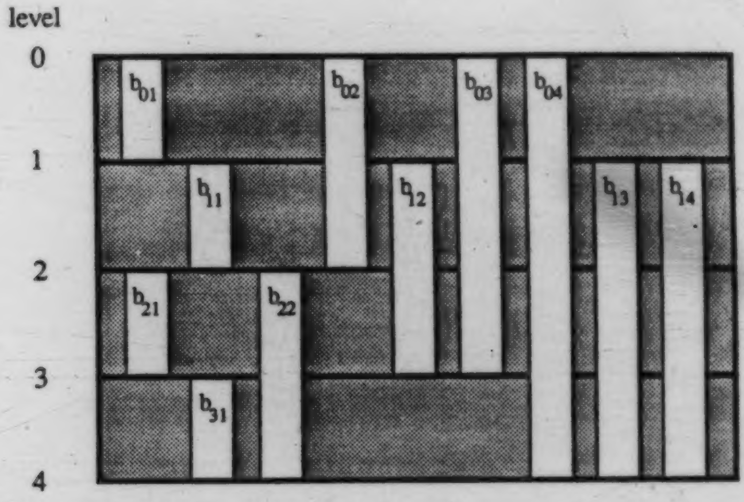


Figure 2.2 A general branching network.

The purpose of the branching network is to provide the potential for adaption in both step size and learning style. For example, the branching network in Figure 2.2 allows a slow student to receive a sequence of four blocks — b_{01}, b_{11}, b_{21}, b_{31}. A bright student may be presented with only one block — b_{04} covering the entire concept. Blocks b_{13} and b_{14} both start and terminate at the same levels, and so can be assumed to differ in instructional style.

Responses to a block question are assigned by response analysis to one of a predetermined number of equivalence classes called answers. Smallwood assumed that, for each branching network, a function v(i,j,k) is defined which is equal to the level of the student after he or she has given the kth answer to the question in block b_{ij}. If the kth answer is completely correct, v(i,j,k) will equal the terminal level of block b_{ij}. Otherwise, v(i,j,k) will equal some level lower than the terminal level. The function v(i,j,k) is fixed by the course author and does not improve with experience[5]. It may aid the reader's understanding to view the network as actually comprised of two components: the set of blocks, which are the nodes of the network, and a set of directed arcs implied by v(i,j,k).

The following control procedure was used to route students through the branching network:

1) Assign the student to level 0.
2) Using some decision process, select for presentation one of the blocks starting at the student's current level.

[After reading information presented in block, student responds to block question.]

3) Classify the response as the kth answer and assign the student to level v(i,j,k).
4) If the student is at the terminal level then halt, else go to step 2.

The only feature which really distinguishes Smallwood's teaching machine from a standard frame-based CAI program is the decision process alluded to in step 2 of the control procedure. It is to this process that we now turn our attention.

In terms of decision theory, the answer k given by a student to the question in block b_{ij} is the **outcome** of deciding to present b_{ij} to the student. One can imagine a goodness or utility value, denoted U_{ijk}, associated with each outcome. Smallwood proposed that the definition of utility be derived from the instructional application. Without knowing with certainty how the student will respond, the machine can only obtain an expected utility E_{ij} for each of the options available to it:

$$E_{ij} = \sum_{k=1}^{K} P_{ijk} U_{ijk} \qquad \text{(equation 2.1)}$$

where P_{ijk} is the probability that the student gives the kth answer in b_{ij}. According to decision theory, utility is maximized over many such decisions if one always selects the option with the maximum expected utility. The problem is now reduced to one of finding good estimates of P_{ijk} and U_{ijk}.

---

[5] Smallwood proposed that future work might find ways to allow the system to determine v(i,j,k).

Smallwood based his estimation of P_{ijk} for student S on the "past history" of S — what would now be called the student model. The student model was a vector of tri-valued elements (RIGHT, WRONG, NOT_TAKEN) representing the student's performance on all questions in the course, including those in the post-test.

The probability of the correct answer is designated by P_{ijk_c}, and its estimate by $\hat{P}_{ijk_c}$. The estimate of P_{ijk} for any k was obtained by first finding $\hat{P}_{ijk_c}$, then partitioning $1-\hat{P}_{ijk_c}$ among the wrong answers according to the distribution of past students answering the question incorrectly. After comparing several approaches to estimating P_{ijk_c}, Smallwood chose one for his empirical study which he referred to as the "intuitive model". It was based on three parameters:

$$\alpha = \frac{\text{number of correct answers in all student models}}{\text{number of questions attempted in all student models}}$$

$$\pi = \frac{\text{number of correct answers by student S}}{\text{number of questions attempted by student S}}$$

$$\phi = \frac{\text{number of students which answered } b_{ij} \text{ correctly}}{\text{number students which attempted } b_{ij}}$$

An intuitive estimate of P_{ijk_c} for student S is given by any function of these parameters which satisfies the following conditions:

- $\hat{P}_{ijk_c} \geq \phi$, $\hat{P}_{ijk_c} = \phi$, $\hat{P}_{ijk_c} \leq \phi$, as $\pi > \alpha$, $\pi = \alpha$, $\pi < \alpha$ respectively. This means that if S is an above-average (average, below-average) student, his expected performance on b_{ij} is better than (equal to, worse than) the average performance of previous students who have attempted b_{ij}.
- $\hat{P}_{ijk_c} \geq \pi$, $\hat{P}_{ijk_c} = \pi$, $\hat{P}_{ijk_c} \leq \pi$, as $\phi > \alpha$, $\phi = \alpha$, $\phi < \alpha$ respectively. This means that if b_{ij} is an easier than average (average, harder than average) question, the expected

performance of S on b_{ij} is better than (equal to, worse than) his own average performance.

- $\hat{P}_{ijk_c}$ increases as π increases.
- $\hat{P}_{ijk_c}$ increases as ϕ increases.

Among the other models considered by Smallwood was one based on multiple regression, with responses to questions already completed serving as predictors. The intuitive model was preferred over others because it allowed for fast computation and incorporated Smallwood's own subjective expectations about the behavior of learners.

A few definitions of utility were suggested, all of which were based on the expected score on the concept post-test. To give an example, suppose one's goal is to maximize the student's score on the post-test. Using the same model as that used to estimate P_{ijk_c}, an estimate can be obtained for P_m, the probability that a student with a particular path through the branching network will give the correct answer for the mth question on the post-test. The utility of the entire path is the expected score on the post-test of any student who has traversed that path:

$$\text{terminal utility} = \frac{\sum_{m=1}^{n} P_m}{n} \qquad \text{(equation 2.2)}$$

where n is the number of questions in the post-test.

The key to Smallwood's method is the recursive definition of utility: the utility of any outcome achieving the terminal level is simply the expected score on the post-test, otherwise the utility of the outcome is the maximum expected utility of all paths continuing from the level achieved by the outcome.

To see this more clearly, one can represent the problem as in Figure 2.3. This representation can be thought of as a game tree, much the same as those used to describe the behavior of chess or checker playing programs. Every time it makes a decision, the teaching machine searches ahead through this tree by following options available at future decision points and evaluating outcomes. Both Smallwood's teaching machine and a chess player engage in a strictly alternating dialogue with an organism whose actions they can only partially predict and control[6]. Typologies of artificial intelligence problems (Rich, 1983) contrast problems of this type with problems in which the environment is completely predictable.

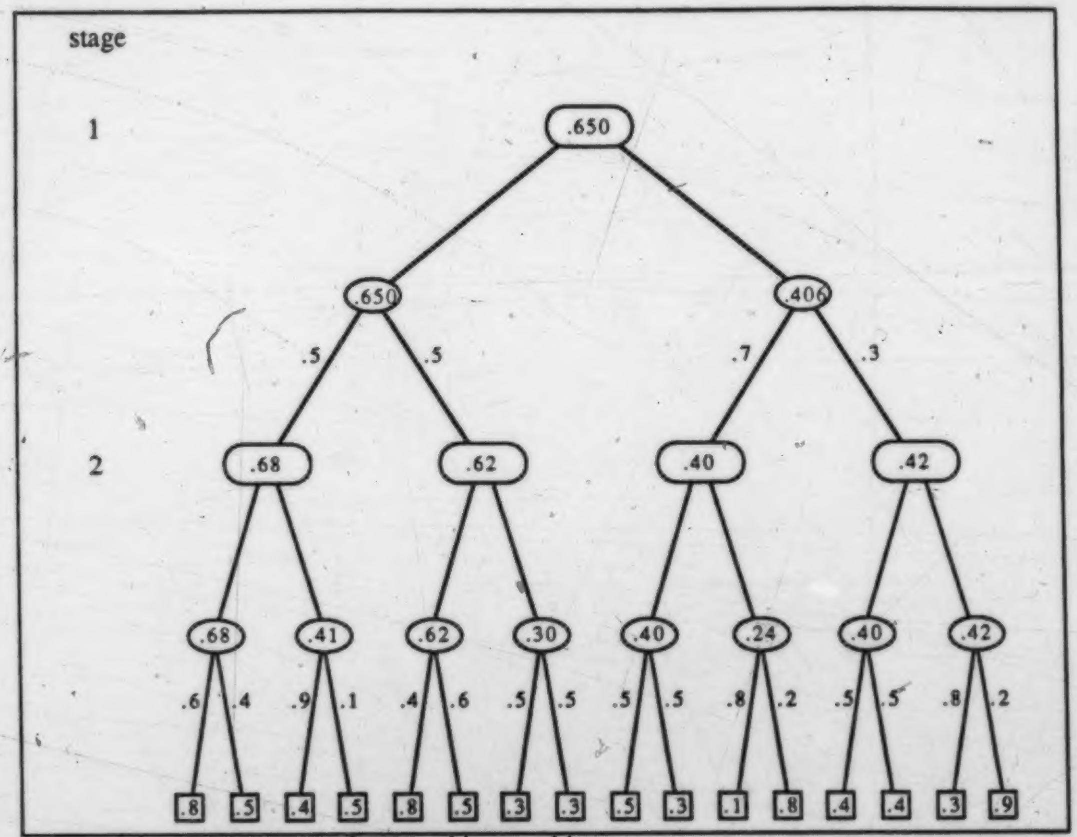

stage
1
2
.650
.650
.406
.5
.5
.7
.3
.68
.62
.40
.42
.68
.41
.62
.30
.40
.24
.40
.42
.6
.4
.9
.1
.4
.6
.5
.5
.5
.5
.8
.2
.5
.5
.8
.2
.8
.5
.4
.5
.8
.5
.3
.3
.5
.3
.1
.8
.4
.4
.3
.9

Figure 2.3 A decision tree for a teaching machine.

Two stages of the decision process are represented in Figure 2.3. Machine decision points appear as capsules and student decision points as ellipses. The terminal nodes in the bottom row contain expected scores (equation 2.2) on the post-test for each path. The arcs below the student decision points represent the answers from which the student may choose. Each answer is labeled with its own probability P_{ijk}. Each student decision point is labeled with the expected utility (equation 2.1) of the machine decision from which it follows. Each machine decision point is labeled with the maximum of the expected utilities of all its available options.

---

[6] Of course, unlike a chess player, a teaching machine is not in "zero-sum" competition with the organism. The consequence is that instead of a single planning strategy which does double duty, the teaching machine requires separate models of teacher and student behavior.

The decision tree in Figure 2.3 is simplified in several respects. All paths shown here are the same length, but in general they will have different lengths. In fact, because the repetition of blocks is allowed, some paths will have infinite lengths. Also, in a more realistic example most decisions would involve more than the two options per decision point depicted here.

Ideally, the teaching machine would start at the current machine decision point, and recurse until it arrived at the end of all paths. Then, as it backed up the tree, it would calculate the values shown as labels in Figure 2.3 by taking the weighted sums (equation 2.1) over all student options and the maximum over all machine options. When it had finally backed up to the original decision point it would choose the option with the maximum expected utility.

In the example, the teaching machine chooses the left branch because it has the greatest expected utility (.65) of the two options available to it. Note that this decision is made even though the path with the greatest expected score on the post-test (.9) belongs to the right sub-tree.

Without some method for dealing with combinatorial explosion, the teaching machine would continue pondering its first decision forever. So Smallwood found it necessary to employ both breadth-wise and depth-wise pruning to limit the search of the decision tree. Both types of pruning introduce error into the decision process.

Breadth-wise pruning was accomplished by dropping from consideration any incomplete path (and therefore the sub-tree dominated by that path) whose estimated probability was less than a predetermined threshold. The estimated probability of a path was presumably calculated as the product of the P_{ijk} encountered so far on the path.

The depth of the search was limited to three stages. In cases where this boundary fell short of the end of a path, an approximation of the expected post-test score was obtained by assuming that the proportion of questions that were answered correctly in getting to the boundary would hold constant for the remainder of the path.

Trying to maximize the post-test score is clearly a poor strategy, because it tends to produce very long paths. The decision criterion actually implemented by Smallwood minimized the learning time necessary for a student to achieve a predetermined expected score on the post-test. Although requiring more complicated calculations, the implemented procedure used a recursive definition of utility and was very similar in other respects to the procedure described here for maximizing the expected score.

There are two major shortcomings to the system presented by Smallwood in his 1962 monograph:

- The branching network should span a partial ordering of subconcepts rather than a linear arrangement of levels. The advantage of assuming a linear arrangement is that the current state of knowledge of the student can be represented very concisely by a single integer. The implication is that the student has learned all and only the subconcepts at or below his current level. However, since Gagne's work on learning hierarchies (Gagne, 1961), it has been apparent that normally a linear arrangement of instructional objectives cannot capture the prerequisite relationships between the objectives. Whenever a linear arrangement is used to approximate a partial order, it is likely that a student set back to a lower level will be re-instructed on material he already knows as he works his way back up to his original position.
- As Smallwood (1970, p. 106) recognized in an article summarizing and revising his views on the optimization of instruction, "the weakest component in that early system was the mathematical model used for the calculation of student response probabilities". The most obvious limitation of that intuitive model was that it could not take advantage of differing degrees of correlation between items (questions). An

item which is highly correlated with the predicted item is allowed no more weight in the prediction than items which are irrelevant. Also, the intuitive model shares with more advanced models (such as item response theory) the often untenable assumption of unidimensionality of test items.

2.6 Atkinson's drill optimization system

Studies on optimal instructional strategies which followed Smallwood's initial work are best exemplified by the adaptive drill programs developed primarily by Atkinson at Stanford (Atkinson, 1972a, 1972b, 1976; Atkinson & Paulson, 1972; Groen & Atkinson, 1966). Atkinson viewed the decision theoretic method, first applied to instruction by Smallwood, as the basis for a theory of instruction. The theory required that the following components be specified for each instructional problem:

- possible states of the learner
- possible instructional actions or operations
- the state transformation resulting from each action
- the cost of each action
- the utility of each state

To make this concrete, a paired-associate learning experiment is described which Atkinson conducted to demonstrate the effectiveness of his method. Items to be learned consisted of German words paired with their English translation. In each trial, the student was faced with the task of supplying the appropriate English word in response to a German word presented as a stimulus. When the student gave a wrong response, the correct English word was supplied as corrective feedback. Each student received 336 trials followed a week later by a post-test covering all 84 items. Students receiving a randomly selected sequence of items averaged 38% on the post-test, those in a learner control condition averaged 58%, and those receiving an optimized sequence averaged 79%. All differences were highly significant.

The student's knowledge of each item was assumed to be in one of three states (P, T, U) between trials. State P indicates that the item has been permanently learned. State T indicates that it has been only temporarily learned and may be forgotten. State U indicates that it has not yet been learned. The effects of a trial (i.e., an instructional action) on knowledge of an item are modeled as two simple Markov processes represented by the two arrays shown below:

$$L(i) = \begin{array}{c} \\ P \\ T \\ U \end{array} \begin{array}{c} \begin{array}{ccc} P & T & U \end{array} \\ \left[\begin{array}{ccc} 1 & 0 & 0 \\ x(i) & 1-x(i) & 0 \\ y(i) & z(i) & 1-y(i)-z(i) \end{array}\right] \end{array}$$

$$F(i) = \begin{array}{c} \\ P \\ T \\ U \end{array} \begin{array}{c} \begin{array}{ccc} P & T & U \end{array} \\ \left[\begin{array}{ccc} 1 & 0 & 0 \\ 0 & 1-f(i) & f(i) \\ 0 & 0 & 1 \end{array}\right] \end{array}$$

L(i) models the learning of item i in a trial presenting item i. F(i) models the forgetting of item i caused by interference from new learning in a trial when some other item is presented. The cell at the rth row and cth column gives the probability that state r will transit to state c. The parameters x(i), y(i), z(i) and f(i) relate to the difficulty and forgetability of item i as determined by prior experimentation. An additional parameter g(i) gives the probability that a specific student is in state P with respect to item i at the start of the drill. These parameters allow for the calculation of a probabilistic statement of the student's knowledge of item i for any given sequence of items.

The parameter g(i) varies with different students and therefore provides the system with the potential for adaption to individual differences. In the experiment described here, the other parameters were fixed for the duration of the study, obviating any potential for self-improvement. However, a system capable of improving its estimates of item difficulty was implemented.

Atkinson wanted to maximize the scores on the delayed post-test. He assumed that items in state T would be forgotten in the week preceding the test, so the definition of utility chosen to guide decision process was the number of items in state P at the end of the instructional session.

A decision process similar to that employed by Smallwood was used to select the next item to maximize the expected utility. Unlike Smallwood's system, all paths through the complete decision tree were of length N. The search process actually implemented was only one stage deep, because Monte Carlo studies indicated that this would obtain a good estimate of the result returned by an N-stage search.

The results of the experiment cannot be taken as evidence that Atkinson's sequence optimization strategy is more effective in a practical instructional situation than a much simpler control strategy or even learner control. It is possible that all the gains in the optimized condition rest on the g(i) parameter. A simple sequencing procedure which allowed the frequency of presentation of item i to be proportional to 1-g(i) may do as well as the optimizing strategy. Unfortunately, Atkinson did not describe how g(i) was obtained.

Although post-test scores are often the only way to measure learning in a practical instructional situation, when the goal set for an optimizing procedure is the maximization of post-test scores, gains produced by the optimal strategy may be illusory. Atkinson's system tended to select items with the greatest probability of transition to state P. This apparently means that for students receiving the optimized sequence, items not recalled in the post-test were the most difficult items in the list. Had a sequel to the experiment been arranged which required students to continue instruction until the entire list was mastered, students in the optimized treatment group may have required as many trials as students in the learner control group.

Atkinson & Paulson (1972) reported on the optimization of frame sequence in the famous Stanford early reading CAI program. The main difference from the procedure already described was that, in this case, the probabilistic learning model did not allow for forgetting. Optimal allocation of instructional time to related drill strands was considered by Chant & Atkinson (1973). This work was based on observations that rate of learning on one strand was often a function of how far the student had progressed along a related strand.

Atkinson's optimization procedure is deserving of more interest and exploration than it has received to date, but its application seems restricted to list drills and other simple tasks for which a credible state transition model can be devised. Smallwood (1970) discussed a hypothetical modification to his teaching machine based on a probabilistic state transition model of the learner. Unfortunately, state transition modeling seems to demand a kind of omniscience of the course author. This was the main objection raised in a critique by Gregg (1970, p. 124):

> The question is how realistic is it to expect that that a precise enumeration of states of the learners and their transition probabilities can be derived from a characterization of the subject matter?

2.7 Adaptive computer-managed instruction

A computer-managed instruction procedure based on ATI principles was described and tested by researchers at Memphis State University (Ross, 1984; Ross & Rakow, 1980, 1981). The aim of the procedure was to prescribe an optimal number of examples for concept learning. The underlying assumption was that the correct number of examples to prescribe prior to instruction is a function of the student's ability as measured by a multiple regression prediction of his post-test score: students with higher predicted achievement should receive fewer examples.

Ross and his associates applied their procedure to the management of printed materials used to teach undergraduate students ten elementary algebra rules. In order to develop the adaptive treatment, considerable prior experimentation was necessary. An initial set of students was given several personality and aptitude tests as well as a test of familiarity with the subject matter. This was followed by instruction on all ten concepts with the students grouped into differing levels of "instructional support" (number of examples). After the instruction phase, a post-test was administered which yielded ten scores for each student. Multiple regression analysis revealed that weighted combinations of four variables, prior familiarity, math reading comprehension, locus of control, and trait anxiety, produced multiple correlations ranging from .53 to .63 across the components of the post-test. Correlations with prior familiarity alone were only slightly lower.

An adaptive program based on this analysis took each student through the following procedure:

- Entry tests are administered yielding measurements of the four predictor variables.
- A predicted score is obtained for each component of the post-test using the distinct set of weights derived for each of the ten concepts. The predicted scores are used to determine the number of examples to prescribe for each concept. For example, one approach was to prescribe six examples for students within 0.5 standard deviations of the mean. Students above this range are prescribed fewer examples (to a minimum of two), and those below receive more (to a maximum of ten). None of the generally available reports of this work provide a justification for the decision rules used, or an account of how they were derived.
- Instruction on each concept with the prescribed number of examples is followed by a concept post-test. If the post-test score is substantially greater or less than predicted, the number of examples prescribed for the following concept is decreased or increased by a commensurate amount.
- A final cumulative post-test is given to serve as a means of comparison with other treatment programs.

Ross & Rakow (1981) reported that students receiving adaptive treatment achieved significantly higher final test scores than students in a learner control group (number of examples chosen by students), a non-adaptive group (fixed number of examples), and a lecture group (fixed number of examples). Ross (1984) also found that adaptive treatment resulted in better achievement than a non-adaptive treatment presenting the maximum number of examples. The most important outcome of this research is that it supports Tobias' view of achievement-treatment interaction as a sound basis for adaptive instruction.

The major shortcoming of the adaptive CMI procedure explored by Ross and his associates, is the ad hoc derivation of rules for determining prescriptions from predicted scores. Although Ross & Rakow (1981) use a different rule than Ross & Rakow (1980), no comparison or evaluation of the merits of these rules is supplied.

The procedure is not mastery-based. One wonders why the student should proceed with a new concept when a test reveals he has not mastered the previous one. It is also not self-improving: no mechanism is considered for updating the regression weights.

2.8 Tennyson's MAIS

The Minnesota Adaptive Instruction System (MAIS) has been described and evaluated in several articles by Robert Tennyson and his co-workers at the University of Minnesota. The goal of MAIS is to determine individualized "instructional treatment prescriptions" from a combination of trait, achievement, and on-task information about the student. In different studies, MAIS has prescribed the number of practice examples needed to learn a concept (Tennyson, 1981; Tennyson & Rothen, 1977) and the period of time the system should wait for a student response before giving away the answer to a question (Tennyson, Park & Christensen, 1985). These studies found evidence suggesting that control by MAIS, and learner control with advisement by MAIS, are more effective than both non-adaptive control and learner control without advisement.

Unlike Ross's adaptive CMI system, the emphasis placed on on-task performance in MAIS has meant that only treatments involving measurable student responses are prescribed. For instance, in teaching several related concepts, an example is presented and the student responds with the concept to which it belongs.

MAIS is based on a Bayesian method for prescribing the minimal number of items to include in a criterion-referenced test developed by Novick and Lewis (1974) for the Individually Prescribed Instruction (IPI) project. In this method the student's ability π is modeled by a beta distribution $B(a,b)$ over the interval between 0 and 1. The user supplies:

- The parameters a and b of a prior distribution representing beliefs about the student's ability before the test. Novick and Lewis suggested that information gleaned from administration of the test to previous students be used. Novick and Jackson (1974) describe interactive programs which allow the user to express the prior distribution in terms of mode and sample size. When applied to prescribing test length, sample size is an estimate of the worth or weight of the prior information in terms of number of test items.
- A criterion ability level π_0 such that students with $\pi \geq \pi_0$ should pass the test (be advanced) and others should fail (be retained).
- A loss ratio $R = p/q$ indicating the relative costs associated with false advancement (p) and false retention (q) decisions. Novick and Lewis indicated that instructional designers in IPI preferred loss ratios ranging from 1.3 to 3.0.

For any prior distribution $B(a,b)$ and test performance where x out of n items are passed, the application of Bayes' theorem generates a posterior distribution $B(a+x,b+n-x)$. A student is advanced if and only if:

$$q\text{Prob}(\pi \geq \pi_0 \mid x, n) \leq p\text{Prob}(\pi < \pi_0 \mid x, n)$$

Under these conditions there is a minimum number of items below which it is impossible to advance a student performing perfectly (x=n) or retain a student who fails all the items (x=0). Also, because n and x may only take on integer values, certain test lengths allow a better approximation of Prob($\pi \geq \pi_0 \mid x, n$) for borderline performances. Novick and Lewis presented tables of recommended test lengths and advancement scores for selected prior distributions, cutting scores, and loss ratios.

MAIS is apparently an attempt to apply the mechanism proposed by Novick and Lewis to the prescription of instructional items rather than test items. Unfortunately, although several published articles refer to MAIS, there is no readily available description giving enough detail to allow for a replication or analytical evaluation of the system. The most complete descriptions can be found in Tennyson and Rothen (1977, 1979) and Rothen and Tennyson (1978), but these leave the careful reader with many unanswered questions.

It is not clear how MAIS uses the prior distribution to select the optimal number of instructional items. There seems to be little justification for simply using the tables presented by Novick and Lewis because a fundamental assumption in their work was that π remains constant, and of course, the purpose of an instructional item is to systematically and permanently change π.

In general, there is no rationale given for significant departures from the Novick and Lewis procedure. For instance, MAIS sets the loss ratio according to the student's score on a prior aptitude test, but Novick and Lewis viewed the loss ratio as a property of the instructional program to be set for all students by the instructional designer.

Tennyson and Rothen (1977) found that MAIS produced higher post-test scores while requiring less learning time than the non-adaptive program. These effects were presented as evidence that MAIS had exploited an aptitude (or achievement) x treatment interaction. However, because the non-adaptive program gave the maximum number of examples (30), a more parsimonious explanation is that subjects in the experimental group benefited by receiving fewer examples regardless of ability. The issue could have been resolved by a replication which included a non-adaptive control group with subjects receiving the average number of examples presented by MAIS. Statistics concerning the number of examples actually presented by MAIS were not reported. No explanation was given for the interesting finding that the standard deviation of learning time was considerably greater in the non-adaptive group.

2.9 Ferguson's branch and test procedure

The remainder of the adaptive instructional systems reviewed in this chapter are primarily concerned, as is this thesis, with exploiting the prerequisite relation between instructional objectives in a learning hierarchy. Although variant forms will soon be encountered, the standard learning hierarchy, as originally conceived by Gagne (Gagne & Paradise, 1961), is an acyclic directed graph with the nodes representing objectives and the arcs representing the prerequisite relation between them. It is interpreted as a conjunctive graph, which means that all prerequisites to an objective should be mastered before the objective is attempted by the student. A more complete definition is presented in Chapter 3.

The Individually Prescribed Instruction (IPI) program in mathematics for grades one through six consists of about 400 objectives divided into 80 modules (Lindvall & Bolvin, 1967). The prerequisite relationships between the objectives in each module are represented by a learning hierarchy. When a student enters an IPI module he is administered a paper and pencil pre-test which determines his mastery or non-mastery of each objective in the module by directly testing each objective.

Ferguson (1970) developed a computerized adaptive testing system for IPI which determined when to terminate testing on a single objective using Wald's sequential probability ratio test[7]. But what makes Ferguson's system of interest here is a branching procedure he incorporated which made a decision about the mastery or non-mastery of every objective after directly testing only a subset.

A complete account of the branching procedure is not available, so the following description is a loose reconstruction based on Ferguson (1970). The procedure requires that the hierarchy first be broken down into linear lists of objectives such that a goal objective and an entry objective be, respectively, at the top and bottom of each list[8].

---

[7] More recent work (Ferguson & Novick, 1973; Kingsbury & Weiss, 1980) has indicated that newer methods based on a combination of item response theory and Bayesian theory are more appropriate for this purpose.

The goal of the branching procedure is to find the highest mastered objective in each list. It is based on the assumption that the hierarchy is valid for every student; which is to say that a position exists on each list below which all objectives are mastered, and above which none are mastered. In practice, some accommodation must be allowed in the algorithm for violations of this principle resulting from fluctuations in student ability during testing, measurement error, flaws in the hierarchy, and so on. Ferguson provides no discussion of the need for, or the nature of, such accommodating mechanisms. But he did report a comparison between the branching procedure and an exhaustive testing of the objectives in one module which found few discrepancies.

The procedure works through each list, moving to the next list when the highest mastered objective has been found. First, an objective in the middle of the list is selected for testing. If the student's probability of mastery is .85 or greater ($p \geq .85$) the objective is assumed to be mastered, otherwise non-mastery is assumed. In the case of mastery, the next objective selected is normally midway between the current objective and the lowest non-mastered objective (or the top of the list), but if the student does very well on the test ($p \geq .93$), the highest untested objective is chosen. In the case of non-mastery, an entirely symmetric strategy was followed in selecting a lower objective for the next test.

When applied to a module containing 18 objectives, the branching procedure was responsible for an average reduction in objectives tested of 61%. The differential treatment of students achieving $p \geq .93$ or $p \leq .43$ was based on an intuitive but, so far, not empirically justified belief that extreme performances predict mastery status of objectives about which the standard learning hierarchy model makes no prediction. This differential treatment is only one of several possible strategies that could be used in trying to optimize selection of the next objective to be tested.

---

[8] In terms of the definitions given in Chapter 3, these lists are paths starting with a goal objective and ending with an entry objective. To prepare for computerized adaption of the branching strategy it was necessary to obtain all such paths, a task which at that time would probably have been done by hand.

2.10 Knowledge-based adaptive sequencing in BIP

Many conventional instructional programs in the pure and applied sciences are built on the reasonable belief that the best way to learn the area is to solve a large number of problems. Unlike programs such as IPI, in which each unit of instruction corresponds to a single objective, the problem-oriented approach demands a many-to-many correspondence between instruction and objectives, and thus introduces an interesting complication to analysis and management of the instructional process.

The BASIC Instructional Program (BIP) developed at Stanford (Atkinson, 1976; Barr, Beard, & Atkinson, 1976) was an attempt to minimize the time necessary to learn fundamental computer programming skills by making the selection of BASIC programming tasks contingent upon previous performance. Like many of the projects applying artificial intelligence techniques to instruction which appeared after Carbonell's ground-breaking SCHOLAR program (Carbonell, 1970), BIP was based on a LISP representation of a semantic net model of the subject matter. In the original version of BIP, the prerequisite relationships were restricted to a linear order.

BIP was capable of presenting to the student about 100 different programming tasks requiring, and thereby teaching, a total of 90 skills (objectives). The number of skills exercised by a single task varied from one to more than ten. The programming tasks, including hints, sample solutions, and skill classifications were human-generated. All solution checking was done by the computer.

BIP maintained a student model containing information about the student's understanding of each skill, as well as a measure of the student's general ability based on his or her success with the tasks. The task selection procedure can be summarized as follows:

1) Search the student model for skills that currently have a high priority. High priority skills are those which were required by a previous task that the student was unable to complete, or those which the student has asked for more work on.
2) When a set of high priority skills has been identified, search the curriculum model for a task that requires some of the high priority skills and which best matches the student's ability.

After comparing BIP with a less flexible procedure which they considered to be representative of "traditional" branching methods, Barr et al. reported that students were able to complete significantly more tasks within a 15 hour course when tasks were selected by BIP. However, a post-test found no significant differences in achievement between the two groups.

In BIP-II (Westcourt, Beard, & Gould, 1977) the prerequisite relation in the curriculum model was extended to allow for a partial ordering of skills (i.e., a learning hierarchy). The task selected by BIP-II was that one which most closely satisfied the following conditions (in the given order):

1) requires no skills with unlearned prerequisites
2) requires m high priority skills
3) requires n unlearned skills

where m and n are certain unspecified functions of the student's demonstrated ability. We are only told that m and n are low when ability is low, and high when ability is high.

2.11 Koffman and Perry's concept selection procedure

Koffman and Perry (1976) described an AIS based on a tree representation of prerequisite relationships between concepts (i.e., objectives) comprising a course on digital systems design. As in BIP, students learned by solving problems, and the system could evaluate student solutions and provide sample solutions if necessary. But Koffman and

Perry's system also included a problem generator that could provide a problem specific to a given concept at a given difficulty level.

The most noteworthy feature of this AIS was a concept selection procedure that tried to predict which next concept should be worked on to advance the student most speedily through the course. The student had the option of accepting the recommendation of the AIS or rejecting it and making his or her own selection. The system then generated one problem relating to the selected concept. The student's response resulted in the increment or decrement of a concept-specific ability variable stored in the student model. It was found that students relied heavily on the system's recommendations at the beginning of the course, but became more independent as the course progressed.

The selection procedure made use of a rather extensive student model. Each student model contained a few general variables (i.e., number of days since last used system, number of concepts worked, number of sessions) and about 17 variables for each concept (e.g., student's ability on concept, number of days since last worked on concept). There were 21 concepts in the course, so each student model must have contained about 360 numeric values.

The concept selection procedure first identified potential concepts for instruction (those permitted by the learning hierarchy), then evaluated the utility of each potential concept by applying a scoring polynomial (a vector of 10 weights) to 10 variables in the student model[9]. The concept with the highest score was recommended. After the student had chosen a concept and worked a problem generated by the system, the student model was updated accordingly.

---

[9] Koffman and Perry cited Samuel's famous checker player (Samuel, 1959) as a precedent for this procedure, but there are important differences. Samuel's program could improve the weights as it played. Also, the polynomial was effective in the checker player because it served as a static evaluation function at the terminal nodes of search of the game tree. But, unlike Smallwood's teaching machine, Koffman and Perry's AIS did no searching.

Weights for the scoring polynomial were derived from a multiple linear regression analysis (r = 0.48) on data gleaned from two semesters of system use. The predicted variable in this analysis was the change in the student's ability on the selected concept. The predictor variables were the 20 variables in the student model relevant to the selected concept. The regression analysis indicated that concepts should be recommended which:

- when last worked resulted in an increased ability.
- the student has a low ability on.
- have not been recently worked on.
- have a large number of prerequisites.

But the most heavily weighted variable suggested that the system should not recommend concepts which have been previously recommended but rejected by the student. The low multiple correlation and the predominance of this latter variable, undermine confidence in the validity of the heuristics outlined above. A refinement of the system, based on a cluster analysis of the data, provided for four separate scoring polynomials. Students were assigned to one of these according to a "compatibility ratio" for which no description was provided.

Koffman's AIS is subject to the same criticism as Atkinson's drill optimization system, to which it bears a resemblance. That is, the system may be greedily recommending the concept expected to result in the greatest immediate advance, but in doing so it is only delaying the time at which the more difficult concepts are dealt with. Therefore, even if one could base decisions on a polynomial derived from a very high correlation, the overall instructional time may be unaffected.

2.12 Heines and O'Shea's rule-based tutor

Heines and O'Shea's (1985) proposal for a rule-based tutorial system for teaching the ReGIS graphics language is one of the clearest applications of Hartley's framework for adaptive instruction. Had it been implemented, their system would have consisted of 1) a task model in the form of a learning hierarchy, 2) a student model giving the student's learning rate and style and the mastery status of each skill (objective) in the task model, and 3) a teacher model comprised of (a) a set of teaching operations for teaching the skills and (b) a set of means-ends guidance rules which were heuristics indicating under which conditions of the student model each of the teaching operations should be triggered.

Even though the proposed tutorial dealt with only a small subset of the ReGIS language, the task model contained about 50 skills. The task model, represented by the production rule formalism, was an augmented variety of learning hierarchy which allowed for both the conjunction and disjunction of prerequisites[10].

The proposed student model was a vector of integer values. It included a mastery status variable for each skill in the task model, a learning rate variable (VERY_FAST, FAST, AVERAGE, SLOW, VERY_SLOW), and a learning style variable indicating the students preference for one of the three types of instruction available (EXPOSITORY, EXERCISE, LABORATORY). The mastery status variable took on a range of values (-3 to +3) reflecting the system's degree of confidence that the student had not mastered or mastered the skill. For example, a status of -3 meant that the student had demonstrated non-mastery in a test, but -1 indicated that non-mastery had only been assumed because the student had demonstrated non-mastery of a prerequisite.

Teaching operations were executed by invoking a LISP function with skill, learning style, and learning rate parameters. Heines and O'Shea did not discuss how the teaching operations were to be prepared or generated, but it becomes an important question when one considers that at least one distinct teaching operation is required for each possible combination of parameter bindings[11].

---

[10] See page 52.

[11] I calculate a minimum of 50 * 3 * 5 = 750 operations for the ReGIS course. However, Heines and O'Shea seem to imply that remedial operations should also be available, bringing the total to 1500.

A summary of a few of the means-ends guidance rules is presented here because they shed light on the design of the student model and teaching operation structures. Also, the last rule given here bears considerable relevance to the project reported in Chapters 4 and 5 of this dissertation.

- If the student is studying a skill for the first time, then call the teaching function with the learning rate and learning style parameters given in the student model.
- If the student is reviewing a skill for which the student model indicates non-mastery, then call the teaching function with learning rate = SLOW and learning style = EXPOSITORY.
- If the student is reviewing a skill for which the student model indicates mastery, then ask the student which learning style is preferred, and invoke the teaching function with the given parameter.
- If the student demonstrates non-mastery on a skill and there is only one prerequisite for the skill on which mastery has not been demonstrated or assumed, then call the teaching function passing that prerequisite as the skill parameter.
- If the student demonstrates non-mastery on a skill and there is more than one prerequisite on which mastery has not been demonstrated or assumed, then call the teaching function passing the prerequisite which is "most likely" lacking. The determination of which prerequisite is most likely lacking is made by some unspecified analysis of other skills sharing some of the same prerequisites.

Heines and O'Shea did not explain whether the AIS or the student is responsible for selecting new skills to study. When a standard conjunctive hierarchy is used, letting the learner choose from the set of skills for which the prerequisites are satisfied is an acceptable, if not optimal, strategy. But when disjunction is allowed in the hierarchy, this shared control strategy can obviously lead to excessively long learning times, because the student may unwittingly choose to study many more skills than are required to achieve the course goals.

2.13 Peachey and McCalla's plan-based CAI

The AIS proposed by Peachey and McCalla (1986) is of special interest here because much of its underlying philosophy also serves as the basis of the work reported in subsequent chapters of this dissertation. They share the view, stated in the introductory chapter, that developing planning and control techniques based on the prerequisite relation, will enable the integration of intelligent tutoring systems into large CAI courses. The purpose of their article was to propose the application of robot planning concepts to instructional planning and execution. The scope of the problem addressed by their AIS was not completely defined, but because details of the application of the system to an example course were presented, the example will be taken as fully representing the characteristics of the problem.

In borrowing principles which arose from the classic AI work on robot planning systems (Fikes & Nilsson, 1971; Sussman, 1975; Sacerdoti, 1977), Peachey and McCalla have extended Hartley's framework for adaptive instructional systems. In their approach, the procedural portion of the teacher model, originally comprised of a flat set of means-ends guidance rules, is subdivided into a planning component and an execution component. The job of the planner is to exploit prior knowledge about global course structure to generate a plan. The job of the executor is to implement the plan and in so doing to respond effectively to any failures which arise.

The form of the student model was a simple list or conjunction of acquired concepts (objectives) and misconceptions. The modeling of misconceptions, a major focus of ICAI (e.g., Burton, 1982), has been conspicuously absent from the other systems reviewed here.

The form of the teaching operators was borrowed directly from STRIPS (Fikes & Nilsson, 1971). Each operator consisted of:

- a name
- a list of preconditions (concepts to be matched with the student model)

- an add list (containing concepts expected to be added to the student model as a result of the operator action)
- a delete list (containing misconceptions expected to be deleted from the student model as a result of the operator action)
- the operator action to be invoked by the executor

Operators which had non-nil delete lists were called remedial operators, because their purpose was to correct misconceptions. Note that, like BIP, there is theoretically a many-to-many mapping of concepts to operators. But in the example course all add lists contained exactly one concept, so the mapping was one-to-many.

The executor, which served as the driver, maintained an ELIGIBLE list containing operators from the plan whose preconditions were true in the student model. The executor arbitrarily chose an operator from the ELIGIBLE list, checked to ensure that the student model did not already contain the concept in the operator add list, applied the operator, and updated the student model according to the student's performance. Only when the ELIGIBLE list was empty, did the executor invoke the planner to generate a new plan on the basis of the current student model, the instructional goal, and the operator set. If the operator set is viewed as implying a learning hierarchy, a plan was essentially an explicit representation of that portion of the learning hierarchy relevant to the current student model. The plan required disjunction to represent the situation where two or more operators had the same add lists. The planner generated a plan by starting with the instructional goal and searching backward through the student model space to the current student model.

Peachey and McCalla's AIS makes much of two principles from robot planners: the separation of reasoning about the problem into planning and execution phases, and the representation in the plan of alternative choices for the executor; but they have failed to make a convincing argument for the application of these principles to the solution of instructional problems. To demonstrate this point, a very simple procedure is presented here which duplicates the behavior of their system on the example they present (with one exception described later), but which conducts no prior search of the student model space. This procedure assumes that the operator form is slightly modified by abandoning the delete list and including misconceptions in the add list by prefixing them with a negation symbol..

```
While the student model does not match the instructional goal do:
    •  If OP list is empty then copy entire operator set to OP list.
    •  Search OP for operator x with preconditions matching student model and add
       list not matching student model.
    •  Delete x from OP.
    •  Apply x.
    •  Update student model according to student's performance.
```

Peachey and McCalla's procedure had the unfortunate characteristic of, whenever a misconception appeared, forcing the application of all eligible non-remedial operators before re-invoking the planner to allow the application of the remedial operator capable of correcting the misconception. The procedure given here may apply a remedial operator immediately, but will not guarantee to do so. Of course, it would be a simple matter to always give remedial operators top priority.

The significant flaw in both procedures is that, because the underlying learning hierarchy is an AND/OR graph, they allow routes which include redundant operators[12]. With this kind of hierarchy, a better approach is to have a planner which finds a single route or sequence of operators which is in some sense optimal, and to re-invoke the planner if and when this plan fails. For example, if an estimated learning time is associated with each operator, the task of such a planner might be to find the route with the minimum total estimated learning time.

---

[12] Readers with access to Peachey and McCalla (1986) can verify that their procedure is capable of selecting the following sequence of operators given the the plan shown on page 90:
COD - DFP - COS1 - SFP - CID - CIS - ED - ES - EQP2 - CEQP.
In this sequence ED and ES are unnecessary. Peachey (personal communication, September 7, 1987) agrees that the executor should not apply redundant operators but maintains that they should be available in the plan.

2.14 Conclusion

Taken as a whole, the work reviewed here, while not comprising an exhaustive account of the area, reveals that the study of adaptive instructional systems lacks a coherent set of goals. Rather than building on previous work, the trend has been for each researcher to design a system based on some method borrowed from decision theory, educational measurement, or artificial intelligence. However, recent emphasis on the prerequisite relation may indicate the emergence of the learning hierarchy as an organizing principle.

Empirical validation of these systems is almost non-existent. The few comparative studies reported are plagued by flaws in method or interpretation.

The student models have been, perhaps necessarily, limited to simple declarative representations. A student model in a typical AIS contains little more than an ordered variable representing general ability, and values representing ability specific to each objective in the course. It is not yet clear how to integrate or interface a domain independent AIS with the relatively well structured student models that have been developed for specific skills (e.g., Brown & VanLehn, 1980).

Most of the work has aimed at adapting either the amount of instruction or practice, or the sequence of objectives. In most cases there was no justification presented for the chosen goal of the adaption process (i.e., what is to be optimized), nor was there a discussion of its psychological basis. For example, when Atkinson reports that an optimized drill produced higher post-test scores than a random drill, we are left casting about for a psychological explanation of the difference, and wondering whether maximizing post-test scores is the best instructional goal.

2.15 References

Atkinson, R. (1968). Computerized instruction and the learning process. *American Psychologist*, 23, 225-239.

Atkinson, R. (1972a). Ingredients for a theory of instruction. *American Psychologist*, 27, 921-931.

Atkinson, R. (1972b). Optimizing the learning of a second-language vocabulary. *Journal of Experimental Psychology*, 96(1), 124-129.

Atkinson, R. (1976). Adaptive instructional systems: Some attempts to optimize the learning process. In D. Klahr (Ed.), *Cognition and instruction*. New York: Wiley.

Atkinson, R., & Paulson, J. (1972). An approach to the psychology of instruction. *Psychological Bulletin*, 78(1), 49-61.

Barr, A., Beard, M., & Atkinson, R. (1976). The computer as a tutorial laboratory: The Stanford BIP project. *International Journal of Man-Machine Studies*, 8, 567-596.

Bracht, G. (1970). Experimental factors relating to aptitude-treatment interactions. *Review of Educational Research*, 40(5), 627-645.

Brown, J., & VanLehn, K. (1980). Repair theory: A generative theory of bugs in procedural skills. *Cognitive Science*, 4, 379-426.

Burton, R. (1982). Diagnosing bugs in a simple procedural skill. In D. Sleeman and J. S. Brown (Eds.) *Intelligent tutoring systems*. London: Academic.

Carbonell, J. (1970). AI in CAI: An artificial-intelligence approach to computer-assisted instruction. *IEEE Transactions on Man-Machine Systems*, 11(4), 190-202.

Chance, V., & Atkinson, R. (1973). Optimal allocation of instructional effort to interrelated learning strands. *Journal of Mathematical Psychology*, 10, 1-25.

Cook, D. (1961). Teaching machine terms: A glossary. *Audiovisual Instruction*, 6, 152-153.

Cronbach, L. (1957). The two disciplines of scientific psychology. *American Psychologist*, 12, 772-775.

Cronbach, L., & Gleser, G. (1957). *Psychological tests and personnel decisions*. Urbana: University of Illinois Press.

Cronbach, L., & Snow, R. (1977). *Aptitudes and instructional methods*. New York: Wiley.

Dewey, J. (1976). The child and the curriculum.In J. Boydston (Ed.) *John Dewey: The middle works 1899-1924* (p. 273-291). Southern Illinois University Press.

Ferguson, R. (1970). A model for computer-assisted criterion-referenced measurement. *Education*, 91, 25-31.

Ferguson, R., & Novick, M. (1973). *Implementation of a Bayesian system for decision analysis in a program of individually prescribed instruction* (ACT Technical Bulletin No. 60). The American College Testing Program, Iowa City.

Fikes, R., & Nilsson, N. (1971). STRIPS: A new approach to the application of theorem proving to problem solving. *Artificial Intelligence*, 2, 189-208.

Gable, A & Page, C. (1980). The use of artificial intelligence techniques in computer-assisted instruction: An overview. *International Journal of Man-Machine Studies*, 12, 259-282.

Gagne, E. (1985). *The cognitive psychology of school learning*. Toronto: Little, Brown.

Gagne, R., & Paradise, N. (1961). Abilities and learning sets in knowledge acquisition. *Psychological Monographs*, 75(14, Whole No. 518).

Glaser, R. (1977). *Adaptive education: Individual diversity and learning*. New York: Holt, Rinehart and Winston.

Glaser, R., & Resnick, L. (1972). Instructional psychology. *Annual Review of Psychology*, 23, 207-276.

Gould, S. J. (1981). *The mismeasure of man*. New York: Norton.

Gregg, L. (1970). Optimal policies or wise choices? A critique of Smallwood's optimization procedure. In W. Holtzman (Ed.), *Computer-assisted instruction, testing, and guidance*. New York: Harper & Row.

Groen, G., & Atkinson, R. (1966). Models for optimizing the learning process. *Psychological Bulletin*, 66(4), 309-320.

Hartley, J. (1972). *The adaptive control of learning tasks* (extended version of B.P.S. lecture). University of Leeds Computer Based Learning Project.

Hartley, J. (1973). The design and evaluation of an adaptive teaching system. *International Journal of Man-Machine Studies*, 5, 421-436.

Hartley, J., & O'Shea, D. (1973). Towards more intelligent teaching systems. *International Journal of Man-Machine Studies*, 5, 215-236.

Heines, J., & O'Shea, T. (1985). The design of a rule-based CAI tutorial. *International Journal of Man-Machine Studies*, 23, 1-25.

Holland, J. (1977). Variables in adaptive decisions in individualized instruction. *Educational Psychologist*, 12, 146-161.

Kingsbury, G., & Weiss, D. (1980). *A comparison of adaptive, sequential, and conventional testing strategies for mastery decisions* (Research Report 80-4). Computerized Adaptive Testing Laboratory, Psychometric Methods Program, Department of Psychology, University of Minnesota.

Koffman, E., & Perry, J. (1976). A model for generative CAI and concept selection. *International Journal of Man-Machine Studies*, 8, 397-410.

Lindvall, C., & Bolvin, J. (1967). Programmed instruction in the schools: An application of programming principles in Individually Prescribed Instruction. *Yearbook of the National Society for the Study of Education*, 66, 217-254.

Merrill, M. D., Schneider, E., & Fletcher, K. (1980). *TICCIT*. Englewood Cliffs NJ: Educational Technology.

Nakayama, K., & Higashibara, Y. (1986). *How to maintain human interaction and individualized learning in a large classroom with microcomputer based CAI*. Paper presented at the World Congress on Education and Technology, Vancouver.

Newell, A., Shaw, J., & Simon, H. (1960). A variety of intelligent learning in a general problem solver. In M. Yovits & S. Cameron (Eds.), *Self-organizing systems* (p. 153-189). New York: Pergamon.

Novick, M., & Jackson, P. (1974). *Statistical methods for educational and psychological research*. New York: McGraw-Hill.

Novick, M., & Lewis, C. (1974). *Prescribing test length for criterion-referenced measurement* (ACT Technical Bulletin No. 18). The American College Testing Program, Iowa City.

Rich, E. (1983). *Artificial intelligence*. New York: McGraw-Hill.

Ross, S. (1984). Matching the lesson to the student: Alternative adaptive designs for individualized learning systems. *Journal of Computer-Based Instruction*, 11(2), 42-48.

Ross, S., & Rakow, E. (1980). Adaptive design strategies for the teacher-managed course. *Journal of Instructional Psychology*, 7, 13-19.

Ross, S., & Rakow, E. (1981). Learner control versus program control as adaptive strategies for selection of instructional support on math rules. *Journal of Educational Psychology*, 73(5), 745-753.

Rothen, W., & Tennyson, R. (1978). Application of Bayes' theory in designing computer-based adaptive instructional strategies. *Educational Psychologist*, 12, 317-323.

Sacerdoti, E. (1977). *A structure for plans and behavior*. New York: Elsevier.

Samuel, A. (1959). Some studies in machine learning using the game of checkers. *IBM Journal of Research and Development*, 3, 211-229.

Skinner, B. F. (1958). Teaching machines. *Science*, 128, 969-977.

Smallwood, R. (1962). *A decision structure for teaching machines*. Cambridge, MA: MIT Press.

Smallwood, R. (1970). Optimal policy regions for computer-directed teaching systems. In W. Holtzman (Ed.), *Computer-assisted instruction, testing, and guidance*. New York: Harper & Row.

Snow, R. (1980). Aptitude, learner control, and adaptive instruction. *Educational Psychologist*, 15(3), 151-158.

Steinberg, E. (1977). Review of student control in computer-assisted instruction. *Journal of Computer-Based Instruction*, 3(3), 84-90.

Sussman, G. (1975). *A computer model of skill acquisition*. New York: Elsevier.

Tennyson, R., & Rothen, W. (1977). Pretask and on-task adaptive design strategies for selecting number of instances in concept acquisition. *Journal of Educational Psychology*, 69(5), 586-592.

Tennyson, R., & Rothen, W. (1979). Management of computer-based instruction: Design of an adaptive control strategy. *Journal of Computer-Based Instruction*, 5(3), 63-71.

Tennyson, C., Tennyson, R., & Rothen, W. (1980). Content structure and instructional control strategies as design variables in concept acquisition. *Journal of Educational Psychology*, 72(4), 499-505.

Tennyson, R. (1981). Use of adaptive information for advisement in learning concepts and rules using computer-assisted instruction. *American Educational Research Journal*, 18(4), 425-438.

Tennyson, R., & Park, O. (1984). Computer-based adaptive instructional systems: A review of empirically based models. *Machine-Mediated Learning*, 1(2), 129-153.

Tennyson, R., Park, O., & Christensen, D. (1986). Adaptive control of learning time and content sequence in concept learning using computer-based instruction. *Journal of Educational Psychology*, 77(4), 481-491.

Tobias, S. (1976). Achievement treatment interactions. *Review of Educational Research*, 46(1), 61-74.

Wenger, E. (1987). *Artificial intelligence and tutoring systems*. Los Altos CA: Kaufmann.

Westcourt, K., Beard, M., & Gould, L. (1977). Knowledge-based adaptive curriculum sequencing for CAI: Application of a network representation. *Proceedings of the Association for Computing Machinery*, Seattle WA, p. 234-239.

Wilkinson, A. (1983). Learning to read in real time. In A. Wilkinson (Ed.) *Classroom computers and cognitive science*. New York: Academic Press.

Chapter 3

A Method for Sequencing Instructional Objectives which Minimizes Memory Load[13]

Since its emergence in the early 1960's, Gagne's learning hierarchy (Gagne, 1962, 1968; Gagne & Paradise, 1961) has been one of the most widely accepted and influential concepts in the field of instructional science. A learning hierarchy consists of a set of instructional objectives (or tasks, or skills) and a set of prerequisite relationships connecting the objectives. Gagne and Briggs (1974, p. 110) noted that "a prerequisite skill is integrally related to the skill which is superordinate to it, in the sense that the latter skill cannot be done if the prerequisite skill is not available to the learner". In practice, because each objective is associated with a module of instructional treatment, the learning hierarchy specifies a partial order for presenting such modules to a student.

Learning hierarchies are normally generated by a subjective process in which the instructional designer starts with goal objectives and recursively breaks them down into prerequisite sub-objectives. This process terminates at sub-objectives assumed to be already mastered by the student population at which the instruction is aimed.

Subject matter experts do not always agree on the partial ordering of a given set of objectives, so several methods, reviewed by Reckase and McKinley (1982), have been proposed for using test data to empirically validate learning hierarchies. Most of these methods (e.g., Dayton & Macready, 1976; Airasian & Bart, 1975; Gagne & Paradise, 1961) are based on the principle that a response pattern in which an objective is passed and the prerequisite of the objective is failed contributes contradictory evidence. Procedures for automatically generating learning hierarchies from test data (Macready, 1975; Bart & Krus, 1973) are based on the same principle.

---

[13] A version of this chapter was published in *Instructional Science* (Nesbit & Hunka, 1987).

Once a valid learning hierarchy has been obtained, it is used to derive a sequence of objectives to be followed by the student. Even when a learner control philosophy prevails, the instructional system should be able to recommend a sequence. Tennyson (1981) found that the achievement of learners allowed to control the amount of instruction was significantly lower than that of learners in a program controlled treatment, unless they were advised on when to terminate instruction on each objective in the lesson. When the objectives are hierarchically related, advice on the best sequence of objectives may have similar value.

The main purpose of this chapter is to propose a criterion for selecting a sequence of objectives from the many sequences which are often permitted by a learning hierarchy. Before describing this criterion and a sequence generation algorithm which exploits it, a brief excursion is taken through some simple graph theory definitions as they apply to learning hierarchies. To provide a practical orientation for this discussion, the sequence generation algorithm, together with certain graph algorithms noted in the following section, can be viewed as forming the first rudimentary facilities for a hypothetical computer assisted instructional design toolkit.

3.1 Learning hierarchies as acyclic digraphs

As other authors (Heines & O'Shea, 1985; Stelzer & Kingsley, 1975) have observed, a learning hierarchy can be formally represented as a directed graph (digraph). A digraph consists of a set of nodes and a set of ordered pairs (u,v), called arcs, where u and v are distinct members of the set of nodes. When a digraph is used to model a learning hierarchy, the nodes represent instructional objectives and the arcs show the prerequisite relationships between the objectives.

Figure 3.1 shows a simple digraph with three nodes: A, B, and C. In interpreting the arcs, which are represented by arrows, we say that A is the parent of B and C, and that B and C are the children of A. If this digraph were representing a learning hierarchy, it would tell us that both B and C are prerequisites of A. In other words, the learning hierarchy can be used to partition the set of 3! linear arrangements or sequences of the three objectives into two mutually exclusive subsets: the set of instructionally valid sequences (BCA, CBA) and the set of instructionally invalid sequences (ABC, ACB, BAC, CAB). An instructionally valid sequence is an ordered list in which all objectives in the hierarchy appear exactly once, and in which every objective occurs at some position after all its prerequisites. Valid sequences are said to be 'permitted' by the learning hierarchy.

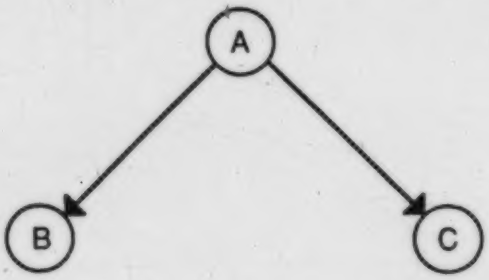


Figure 3.1 A simple digraph.

The convention followed here is that arcs point toward prerequisites. The opposite direction may seem more natural, as it is suggestive of the movement of students through the hierarchy, but it is less convenient for defining certain types of learning hierarchies and for describing the algorithms presented in this chapter.

With learning hierarchies, the arrowhead can be dropped if we adopt the convention that prerequisites always appear lower on the page than their parents. Figure 3.2 shows a learning hierarchy cited by Case and Bereiter (1984). In digraphs of this form the number of arcs exiting downward from a node is the outdegree of the node. The number of arcs entering the node from above is the indegree of the node. For example, node A in Figure 3.2 has indegree zero and outdegree three. In learning hierarchies, nodes with indegree zero represent goal objectives and those with outdegree zero represent entry objectives.

A path (not to be confused with a sequence) is a list of one or more nodes, starting with any node in the digraph, such that each succeeding member of the list is a child of its predecessor in the list. The length of a path is n-1, where n is the number of nodes in the path. A trivial path is a list containing only one node, and therefore having a length of zero.

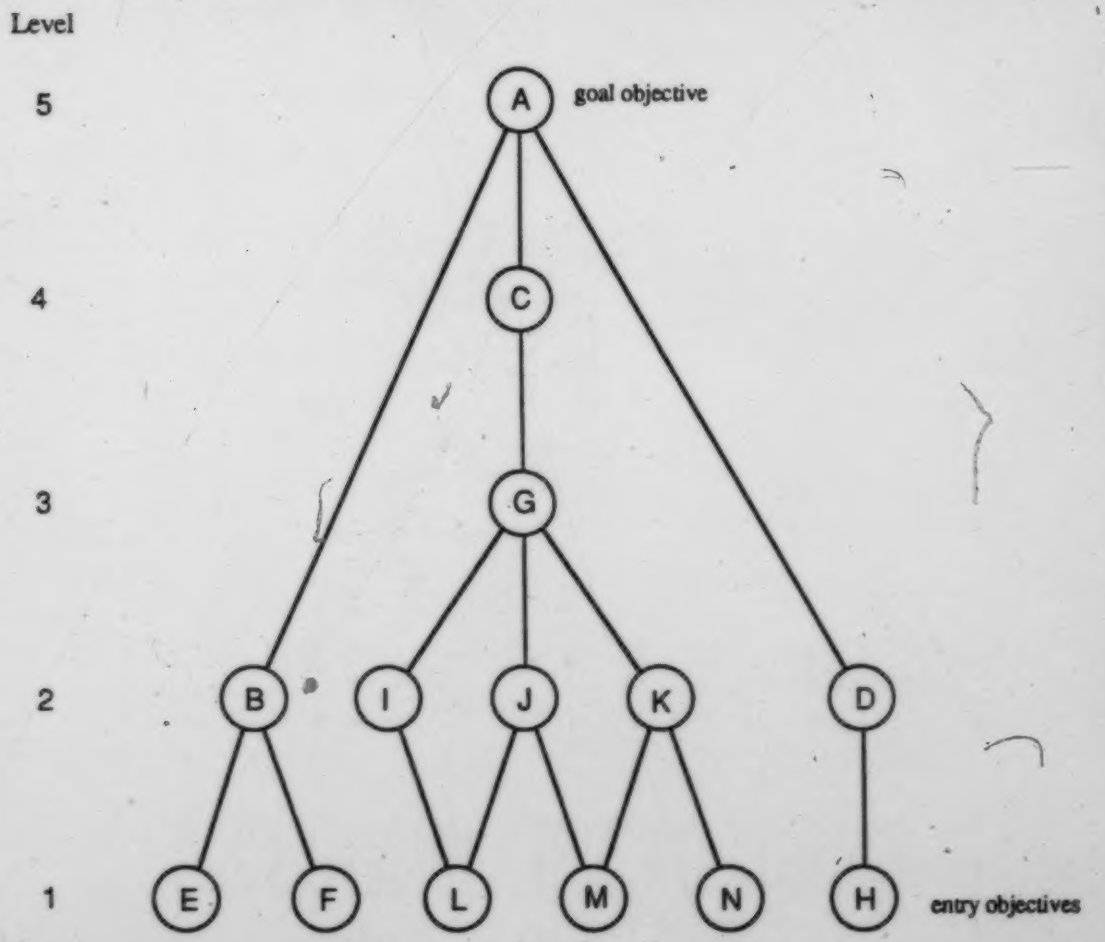


Figure 3.2 A learning hierarchy.

A digraph is cyclic if and only if it has one or more non-trivial paths which begin and end with the same node. Implicit in discussions by Gagne and others, is the tenet that learning hierarchies are acyclic. Incidentally, the absence of cycles ensures that there will be at least one entry objective and at least one goal objective. So perhaps a graph algorithm capable of checking whether a learning hierarchy entered by a course designer is free of cycles, should be the first to go into a toolkit for computer assisted instructional design. A simple algorithm which tests for the acyclic property is described by Robinson and Foulds (1980, p. 73-75).

3.1.1 Level numbers

Sometimes it is useful to assign a positive integer to each instructional objective indicating its level in the hierarchy: a frequently encountered application being the numbering of university courses (e.g., the high order digit in "Psych 100"). In graph-theoretic terms the level number of a node in an acyclic digraph may be defined as the length of the longest path of which it is the first node. Entry objectives are the last node of every path to which they belong, so every path beginning with an entry objective is trivial and has a length of zero. Therefore entry objectives are always assigned to level zero by the given definition. It may be preferable to increment all level numbers by one so that the lowest level is one rather than zero. Figure 3.2 illustrates level assignment by this method.

3.1.2 Inreach and outreach

If *u* and *v* are nodes in a digraph, and there exists a path from *u* to *v*, then we say that *v* is reachable from *u*. The inreach of a node *v* is the set of all nodes from which *v* is reachable, including *v* itself. The outreach of a node *u* is the set of all nodes reachable from *u*, including *u* itself. So the outreach of an objective *u* in a learning hierarchy is the set of all objectives in the hierarchy which must be taken before *u*, plus *u* itself.

3.1.3 Interpretation and treatment of inessential arcs

If (*u*,*v*) is an arc in an acyclic digraph, then (*u*,*v*) is inessential if there is a path from *u* to *v* which does not traverse (*u*,*v*), and essential if there is no such path. Figure 3.3 contains an example of an inessential arc. A prerequisite relationship represented by an inessential arc has no effect on the set of sequences permitted by the learning hierarchy. Therefore, if the only purpose of the hierarchy is to define this set, an algorithm which simplified the hierarchy by detecting and deleting inessential arcs (Robinson & Foulds, p. 85) would be a useful addition to the toolkit.

Prerequisite relationships are also used to implicate sub-objectives as causes of failure when a student is unable to master an objective. If we assume, as Gagne and Briggs seem to, that mastery of an objective implies mastery of all objectives within its outreach, then inessential arcs are indeed redundant. However, if prerequisites are allowed in the hierarchy which are necessary for learning the new objective, but which are not incorporated and practiced as part of it, then "inessential" arcs may represent information valuable to the diagnostic process. It may be necessary to enable the author to specify both types of relationships (let us call them integral and non-integral relationships), and to only allow an inessential arc when it does not short-cut a path connected by arcs of the integral kind.

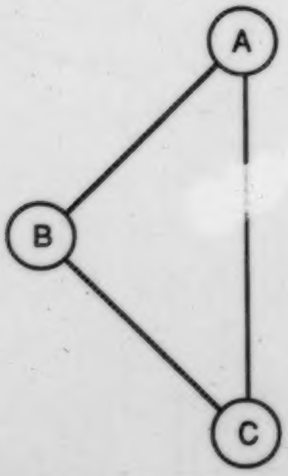


Figure 3.3 The arc from A to C is inessential.

3.1.4 Augmented learning hierarchies

As Gagne (1968) observed, a standard learning hierarchy cannot represent alternative ways of achieving an instructional goal. One hypothesis accounting for the contradictory fail-pass response pattern (in which a prerequisite is failed but its parent is passed) which can thwart the validation of a learning hierarchy, is that a known prerequisite is sufficient but not necessary and that students producing this pattern had mastery of another sufficient prerequisite not represented in the model. This limitation can be overcome by a representation, used both in the task models of Heines and O'Shea (1985) and in Pask's entailment structures (Pask, Kallikourdis, & Scott, 1975), which would indicate that any one of a specified set of sub-objectives can serve as a prerequisite. In other words, hierarchies augmented by such a representation are AND/OR graphs, allowing for both conjunction and disjunction of prerequisites where standard Gagne hierarchies allow only conjunction. The work reported here deals only with standard conjunctive hierarchies.

3.2 Memory load

The set of sequences of instructional objectives permitted by a learning hierarchy can be surprisingly large. For example, the learning hierarchy in Figure 3.2 is a partial ordering of only 14 objectives, yet it allows about 1.6 million different sequences. In fact, the number of permitted sequences is often so large that computer programs attempting to count them by exhaustive search will not terminate within a reasonable period of time. Are there criteria available for selecting the most instructionally effective sequence from this large set?

Posner and Strike (1976) reviewed and categorized many of the principles for sequencing instructional content which have been proposed in the last eighty years. Although the learning hierarchy principle has dominated in recent years, other principles worthy of consideration include the ordering of objectives from most familiar to least familiar, from least difficult to most difficult, from most interesting to least interesting, and so on. Several different sequencing principles have been used in ICAI programs (Wenger, 1987) . If the learning hierarchy is given top priority, other principles can still be invoked within the constraints it imposes. Gagne and Briggs (1974) suggest that resource availability determine sequencing after the constraints of the learning hierarchy have been satisfied. This chapter proposes a new sequencing principle which may be viewed as an extension of the learning hierarchy principle.

Although all permitted sequences ensure that when the student begins to learn a new objective the prerequisites will have been mastered at some previous time, it is possible that some or all of the prerequisites will have been forgotten. The probability of forgetting is known to increase with time as a result of interference from other learning. Therefore, one approach to finding a "best" sequence is to minimize the instructional time elapsing between when an objective is learned and when it is needed for further learning.

Memory load is an attempt to provide, for the practical purposes of the instructional designer, a relative estimate of the retroactive inhibition effects contributing to the forgetting of prerequisites in a sequence with an underlying hierarchical structure. The term memory load was chosen because sequences subject to greater interference presumably place a greater burden on the memory ability of the learner.

Suppose an estimated learning time t is associated with each objective. A memory load value can be obtained by any permitted sequence of objectives:

$$\text{memory load} = \sum_{i=1}^{m} \sum_{j=1}^{q_i} t_{ij} \qquad \text{(equation 3.1)}$$

where m is the number of arcs in the hierarchy, q_i is the number of objectives intervening between the parent and child of the ith arc, and t_{ij} is the estimated learning time of the jth objective intervening between the parent and child of the ith arc.

The following sequence is permitted by the hierarchy in Figure 3.2:

N M K L J I G C F E B H D A

Assuming that all objectives have a learning time of 1.0, the memory load for this sequence is 16.0, which happens to be the minimum memory load for the hierarchy. The arc from A to C is stretched by five intervening objectives, so it contributes 5.0 to the memory load. The arc from D to H has no intervening objectives, so it contributes 0.0 to the memory load.

3.3 Deficiencies of memory load as a model of forgetting

The utility of minimizing memory load in the instructional design process depends on the accuracy of memory load as a predictor of forgetting. Four potential sources of error are apparent:

1. Time spent in activities external to the learning hierarchy is a cause of forgetting, but is not captured by the memory load model. So one expects that the accuracy of memory load as a predictor of forgetting will vary with the degree of intrusion of external activities at course delivery time.
2. Some objectives are simply more memorable than others, but the memory load model fails to differentiate between them. Many psychological factors will come into play to vary the likelihood that a specific objective will be remembered by a specific individual. For example, objectives which the student finds interesting are likely to be remembered longer than those which are boring. If an objective is known to be relatively resistant to forgetting, one can afford to allow it greater separation from the objectives having it as a prerequisite in order to shorten the time spanned by more sensitive prerequisite relationships.
3. The probability of forgetting usually increases as a non-linear function of time, but the memory load model assumes linearity. One expects intervening objectives to vary in the strength with which they interfere with memory for the prerequisite. It is conceivable that some intervening objectives may help the student to recall the prerequisite (negative interference). Therefore, one cannot even be sure that the probability of forgetting will be a monotonically increasing function of instructional time.
4. The definition of memory load given by equation 3.1 is really the total memory load associated with a sequence, not the memory load currently experienced by a student at a particular point in the sequence. As the student progresses through the sequence this current memory load may increase or decrease. No consideration is given here

to the distribution of total memory load over the sequence, but this may turn out to be an important factor. For example, if we assume that the student has a memory load threshold above which forgetting starts to occur, then it is plausible that objectives arranged in a sequence with a uniform distribution of memory load would be learned more easily than the same objectives arranged with an equal memory load having a peaked distribution.

These deficiencies suggest that a more accurate model could be constructed which would include as parameters some of the factors ignored by the current model. Unfortunately though, the additional information required by such a model is usually unavailable to the instructional designer. However, there is reason to expect that when the instruction is delivered by computer, the frequency with which each prerequisite is forgotten could be recorded and fed back into the sequence planning process.

3.4 A sequencing algorithm which minimizes memory load

Consider the problem of finding any sequence permitted by a given learning hierarchy such that the sequence has the minimum memory load. The huge number of sequences permitted by many learning hierarchies severely limits the usefulness of methods relying on an exhaustive search of the sequence space. A preliminary study which applied A* search[14] using a heuristic based on the minimum possible memory load associated with an objective not yet included in the sequence, found that even this approach is frequently defeated by combinatorial explosion.

The ideal solution would be a more direct one which would avoid searching the sequence space altogether. Unfortunately, no algorithm of this type has been found which succeeds with learning hierarchies in general[15]. However, a simple and efficient algorithm is presented here which is conjectured to generate a minimum memory load sequence for a subclass of learning hierarchies called learning trees.

---

[14] See Barr and Feigenbaum (1981, p. 64) for an introduction to A*.

[15] Even and Shiloah (1975) proved that this problem, the optimal arrangement of nodes in an acyclic digraph, is NP-Complete. NP-Complete problems are those that have no known algorithm which will always provide a solution in polynomial time, and are equivalent to other NP-Complete problems in the

A tree is an acyclic digraph containing only nodes having indegrees less than or equal to one, and exactly one node having indegree zero. Any learning hierarchy satisfying these restrictions is a learning tree. It turns out that learning trees are a fairly frequent form of learning hierarchy. An informal survey of 49 learning hierarchies cited in 14 documents (Briggs, 1972; Briggs & Wager, 1981; Case & Bereiter, 1984; Dick & Carey, 1978; Edney, 1972; Gagne, 1962, 1965, 1968; Gagne & Briggs, 1974; Gagne & Paradise, 1961; Heines & O'Shea, 1985; Riban, 1969; Walter, 1965; Wollmer & Bond, 1975) found that 31% were learning trees, 59% were non-trees having only one goal objective, and 10% had multiple goal objectives.

Table 3.1 The GENERATOR algorithm

1) To each node (objective) in the learning tree assign the value T, the sum of the estimated learning times (t) of all nodes in its outreach.

2) Initialize the sequence to be a null string of nodes.

3) Invoke the following recursive procedure SUB_GENERATOR passing it the goal node of the learning tree.

```
procedure SUB_GENERATOR ($\alpha$ : node);
var $\beta$ : node;
begin
while $\alpha$ has any unmarked child do
    begin
    $\beta \leftarrow$ unmarked child of $\alpha$ with greatest T;
    SUB_GENERATOR ($\beta$);
    end;
append $\alpha$ to sequence;
mark $\alpha$;
end;
```

---

sense that if an efficient algorithm is ever found for just one NP-Complete problem, efficient algorithms for all the others could be immediately derived (Garey & Johnson, 1979).

GENERATOR, an algorithm which finds a minimum memory load sequence permitted by a learning tree, is given in Table 3.1. When applied to learning trees, the effect of the algorithm is to find a permitted sequence of objectives with a) the objectives belonging to the same outreach positioned contiguously, and b) the prerequisites of any objective ordered such that a prerequisite having an outreach with a greater total learning time will precede one having a lesser total learning time. Steps 1 and 3 are essentially depth-first searches of the learning tree, so the time complexity of GENERATOR is O(*e*), where *e* is the number of arcs in the tree.

3.5 Generalizing the algorithm to deal with non-trees

When no procedure is known for obtaining an optimal solution to a problem, one is often forced to rely on methods that obtain solutions which are at least better than those resulting from an unguided search. In the hope of obtaining such a method, a version of GENERATOR was developed which differs from the version given previously in only one respect: the node β, passed in the recursive call to SUB_GENERATOR, is the unmarked child of α having the greatest value T_k/P_k, where node k is any unmarked child of α, T_k is the sum of the estimated learning times of all nodes in the outreach of node k, and P_k is the number of unmarked nodes *not in the outreach of k* which are parents of nodes that are in the outreach of k. It should be clear that $P_k \geq 1$ because the parents of any node k cannot be in the outreach of k, and cannot be marked if k is not marked. In the case of trees, $P_k = 1$ for all k, so the algorithm reduces to its original form.

Consider the application of the modified version of GENERATOR to the example in Figure 3.4. The nodes are labeled with their T values and initial P values. The sequence generated by the algorithm is:

EDCBA.

Note that, unlike the original version of GENERATOR, the modified version is constrained to choosing E rather than D as the initial node because $T_E/P_E > T_D/P_D$.

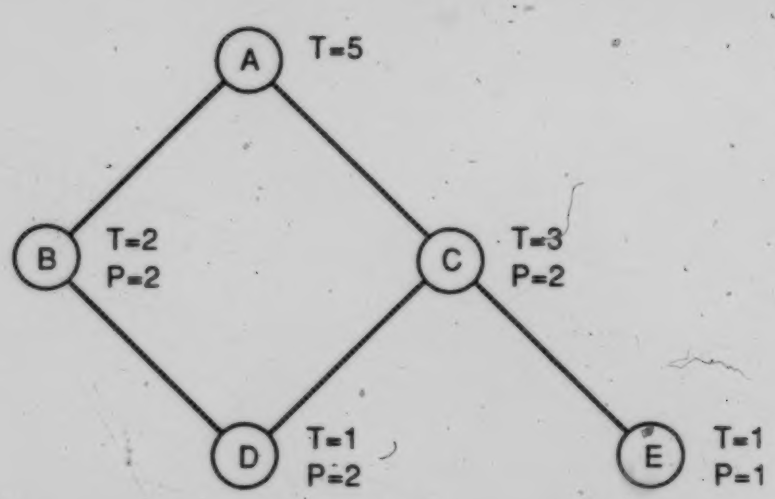


Figure 3.4 A hierarchy with nodes labeled by GENERATOR.

Tests were conducted with several published examples of learning hierarchies which are not trees to see whether the modified version of GENERATOR holds promise as a heuristic method. The results of the tests are summarized in Table 3.2. Because it is only hierarchies permitting a large number of sequences which pose a problem, the hierarchies with the largest number of objectives of those hierarchies surveyed were chosen for testing. With the exception of hierarchy 15 (H15), which had one of its two goal objectives deleted, only hierarchies having exactly one goal objective were used. Estimated learning times are not usually given with published examples of hierarchies, so objectives were simply assigned a learning time of one (except for those in H3 and H19 which were assigned random learning times).

All programs were written in Pascal and executed on a Digital Equipment Corporation VAX 11/780 minicomputer. On this machine, GENERATOR consumed less than 100 milliseconds of CPU time per hierarchy.

Entries in the "no. of sequences", "max ML", "ave ML", and "min ML" columns were obtained by a depth-first exhaustive search program which counted sequences and kept track of the maximum, average, and minimum memory loads encountered. An arbitrary upper limit of 12 hours was set on the CPU time required by the depth-first search. In cases where this search was not completed in the allotted time, the accumulated count is presented as a lower bound on the number of permitted sequences, and a random search program was run for another 12 hours to get more accurate memory load estimates.

A second phase for the sequencing algorithm was developed, called SHIFTER, which incrementally improves a sequence produced by GENERATOR. SHIFTER steps through the sequence, and at each node tests whether the node can be moved to any new position such that memory load is decreased. SHIFTER only terminates when no single node can be moved to improve the memory load. Like other hill-descending methods, SHIFTER gets snagged on local minima.

Table 3.2 Performance of GENERATOR.

Hierarchy (H)	no. of nodes	no. of sequences	max ML	ave ML	min ML	GENERATOR ML	SHIFTER ML
1) Gagne & Briggs (1974, p. 117)	15	90090	29	22.0	9	9	9
2) Edney (1972, p. 103)	29	≥1790000	≥270	241.1	≤122	67	67
3) Edney (1972, p. 103)	29	≥1790000	≥17752	12403.4	≤6883	2733	2733
4) Gagne (1962, p. 359)	10	756	22	17.0	9	9	9
5) Gagne & Briggs (1974, p. 118)	9	480	24	18.2	10	10	10
6) Gagne & Briggs (1974, p. 114)	11	480	24	19.8	15	15	15
7) Riban (1969, p.119)	11	16800	35	26.4	12	12	12
8) Gagne & Briggs (1974, p. 116)	12	44	15	13.3	11	11	11
9) Case & Bereiter (1984, p. 145)	14	1570140	57	43.5	16	16	16
10) Dick & Carey (1978, p. 29)	9	52	12	10.8	9	10	9
11) Walter (1965, p. 52)	16	16200	40	31.6	20	21	20
12) Wollmer & Bond (1975, p. 8)	16	6336	53	45.5	39	41	39
13) Gagne (1965, p. 181)	18	33880	83	72.8	60	64	60
14) Dick & Carey (1978, p.46)	18	≥3990000	≥71	52.7	≤23	23	23
15) Gagne & Paradise (1961, p. 6)	23	≥2700000	≥149	112.3	≤64	46	46
16) Briggs (1972, p. 121)	23	≥2680000	≥104	74.8	≤44	44	44
17) Walter (1965, p. 49)	20	≥3440000	≥134	106.0	≤82	96	82
18) Gagne (1965, p. 150)	20	≥3550000	≥147	129.2	≤98	124	105
19) Dick & Carey (1978, p. 46)	18	≥3990000	≥3906	2743.9	≤1176	1171	1171

H1 through H3 are the only learning trees in the sample. They were included to demonstrate the effectiveness of GENERATOR when applied to hierarchies of this type. The search program was able to complete an exhaustive search of the sequence space of H1, and a comparison of columns reveals that GENERATOR did produce a minimum sequence. In the case of H2, GENERATOR produced a sequence having a memory load much lower than that of the best sequence found by the random search program. H3 is the same hierarchy as H2, but with randomly assigned learning times (integers between 1 and 100).

H4 through H9 are hierarchies that could be thoroughly searched and for which GENERATOR produced minimum sequences. H10 through H13 are those that could be thoroughly searched but for which GENERATOR did not produce a best sequence. However, in these four cases SHIFTER was able to improve the generated sequences to obtain minimum sequences.

H14 through H19 are hierarchies whose sequence space could not be thoroughly searched within the allotted time. H19 is the same hierarchy as H14 except that, like H3, it was assigned random learning times. H15 and H18 are particularly noteworthy: the former because its result was considerably better than that of the search, and the latter because its result was considerably worse.

To summarize the results in Table 3.1, for 9 of the 15 distinct non-tree hierarchies GENERATOR produced a sequence as good as the best found by a depth-first or random search of the sequence space running for up to 12 hours of CPU time. For 5 of the remaining 6 hierarchies, SHIFTER was able to improve the sequence produced by GENERATOR to obtain a sequence as good as that found by the search programs. The sequence produced by GENERATOR was under the estimated average memory load for the hierarchy in all cases.

3.6 Conclusion

The evidence seems to support the hypothesis that the generalized version of GENERATOR is useful for finding sequences with low memory loads when given learning hierarchies with a single goal objective. With what is presently known, perhaps the best strategy for sequencing objectives when one has allotted a fixed amount of CPU time for the task is to first obtain a sequence from GENERATOR, improve it with SHIFTER, then spend the remaining time randomly searching the sequence space. Whenever a better sequence is found an attempt should be made to improve it with SHIFTER. Although it is slower and cannot examine as many sequences, random search is preferable to an ordered depth-first traversal because it is not localized to one region of the sequence space, and the sequences it sees will have a wider range of memory loads.

There are several problems which might be included on an agenda of future research in the area. One essential but arduous enterprise will be empirically validating the utility of memory load as a criterion for sequencing instructional objectives. Several studies, involving instructional treatments covering various subject areas, will be required before a convincing conclusion emerges.

When memory load is viewed as a model of forgetting in the instructional process, inherent sources of error become evident. By including relevant information that can be known or estimated at course design time, it is possible that a better model could be developed which is still usable as a tool for instructional planning. Throughout this chapter, the relation between instructional objectives and their surface manifestations which are presented to the learner has been assumed to be one-to-one. However, there is no essential incompatibility between the approach followed here and systems which allow a many-to-many relation between objectives and the instructional modules presented to the student (e.g., Smallwood, 1962; Westcourt, Beard, & Gould, 1977).

There is room for more work on algorithms for finding minimum memory load sequences. A major disadvantage of GENERATOR is that it cannot plan the remainder of a partially completed sequence. An algorithm capable of finding a low memory load completion of a partial sequence could be used at course delivery time to fit the instructional plan to the current state of the student model.

3.7 References

Airasian, P., & Bart, W. (1975). Validating a priori instructional hierarchies. *Journal of Educational Measurement*, 12, 163-173.

Barr, A., & Feigenbaum, E. (1981). *The Handbook of Artificial Intelligence* (Vol. 1). Los Altos, CA: Kaufmann.

Bart, W., & Krus, D. (1973). An ordering theoretic method to determine hierarchies among items. *Educational and Psychological Measurement*, 33, 291-300.

Briggs, L. (1972). *Student's Guide to Handbook of Procedures for the Design of Instruction*. American Institutes for Research.

Briggs, L., & Wager, W. (1981). *Handbook of Procedures for the Design of Instruction*. Englewood Cliffs, NJ: Educational Technology Publications.

Case, R., & Bereiter, C. (1984). From behaviorism to cognitive behaviorism to cognitive development: steps in the evolution of instructional design. *Instructional Science*, 13, 141-158.

Dayton, C., & Macready, G. (1976). A probabilistic model for validation of behavior hierarchies. *Psychometrica*, 41, 189-204.

Dick, W., & Carey, L. (1978). *The Systematic Design of Instruction*. Glenview, IL: Scott, Foresman & Company.

Edney, P. (1972). *A Systems Analysis of Training*. London: Pitman.

Even, S., & Shiloah, Y. (1975). *NP-Completeness of Several Arrangement Problems* . (Department of Computer Science, Technical Report #43). Haifa, Israel: Technion Institute.

Gagne, R. (1962). The acquisition of knowledge. *Psychological Review*, 69(4), 355-365.

Gagne, R. (1965). *The Conditions of Learning*. New York: Holt, Rinehart, & Winston.

Gagne, R. (1968). Learning Hierarchies. *Educational Psychologist*, 6(1), 1-9.

Gagne, R., & Briggs, L. (1974). *Principles of Instructional Design*. New York: Holt, Rinehart, & Winston.

Gagne, R., & Paradise, N. (1961). Abilities and Learning Sets in Knowledge Acquisition. *Psychological Monographs*, 75(14, Whole No. 518).

Garey, M., & Johnson, D. (1979). *Computers and Intractability: A Guide to the Theory of NP-Completeness*. San Francisco: Freeman.

Heines, J., & O'Shea, T. (1985). The design of a rule-based CAI tutorial. *International Journal of Man-Machine Studies*, 23, 1-25.

Macready, G. (1975). The structure of domain hierarchies found within a domain referenced testing system. *Educational and Psychological Measurement*, 35, 583-598.

Nesbit, J., & Hunka, S. (1987). A method for sequencing instructional objectives which minimizes memory load. *Instructional Science*, 16, 137-150.

Pask, G., Kallikourdis, D., & Scott, B. (1975). The representation of knowables. *International Journal of Man-Machine Studies*, 7,15-134.

Posner, J., & Strike, K. (1976). A categorization scheme for principles of sequencing content. *Review of Educational Research*, 46(4), 665-689.

Reckase, M., & McKinley, R. (1982). *The Validation of Learning Hierarchies* (Research Report ONR 82-2). Iowa City: The American College Testing Program.

Riban, D. (1969). *An investigation of the relationship of Gagne's hierarchical sequence model in mathematics to the learning of high school physics*. Doctoral dissertation, Purdue University (University Microfilms No. 70-8957).

Robinson, D., & Foulds, L. (1980). *Digraphs: Theory and Techniques*. London: Gordon & Breach.

Smallwood, R. (1962). *A Decision Structure for Teaching Machines*. Cambridge, MA: MIT Press.

Stelzer, J., & Kingsley, E. (1975). Axiomatics as a paradigm for structuring subject matter. *Instructional Science*, 3, 383-450.

Tennyson, R. (1981). Use of adaptive information for advisement in learning concepts and rules using computer-assisted instruction. *American Educational Research Journal*, 18(4), 425-438.

Walter, K. (1965). *Authoring individualized learning modules: A teacher training manual* (Title III, E.S.E.A). Kensington, MD: Project Reflect.

Wenger, E. (1987). *Artificial intelligence and tutoring systems: Computational and cognitive approaches to the communication of knowledge*. Los Altos CA: Kaufmann.

Westcourt, K., Beard, M., & Gould, L. (1977). Knowledge-based adaptive curriculum sequencing for CAI: Application of a network representation. *Proceedings of the Annual Conference of the Association for Computing Machinery*, Seattle, WA, October, p. 234-239.

Wollmer, R., & Bond, N. (1975). *Evaluation of a Markov-decision model for instructional sequence optimization* (ARPA Order No. 2284). Los Angeles: University of Southern California.

Chapter 4

Inductive Learning and Adaptive Instruction[16]

In addition to their use in the sequencing of objectives, learning hierarchies can serve a valuable diagnostic function. When the student exhibits difficulty in learning an objective, the learning hierarchy points to prerequisite objectives which may have been forgotten or never mastered (Heines & O'Shea, 1985). In this case the instructional system can either review all the prerequisites, or it can try to be more adaptive by identifying which of the implicated prerequisites are actually in a state of non-mastery. The cost of the adaptive strategy is the time and tedium of the testing required to determine the student's understanding of each of the implicated prerequisites.

The major premise of the project reported here is that prior information about the student can be used by a set of classification rules to reduce the amount of required testing. Course authors cannot always be expected to provide an accurate set of classification rules at the time a course is being designed, so an inductive learning procedure (ILP) is presented in this chapter which refines the rules given by the author, or learns a set of rules from scratch if none are provided.

ILP can be viewed as replacing the ad hoc conditional branching that authors are supposed to specify in conventional CAI programs (e.g., Feingold, 1968) with an empirically justified decision process. When an author uses an authoring system to specify a conditional branch, he is stating a hypothesis about the class of students requiring review of, or introduction to, a particular objective. For example, the following conditional branching statement represents the hypothesis that the students who should review unit_x are those who have a response time greater than ten seconds and have not been in unit_x within the last four days:

---

[16] A condensation of chapters 4 and 5 has been submitted for presentation at the International Conference on Intelligent Tutoring Systems, Montreal, June, 1988.

```
IF response_time > 10 sec AND time_since_unit_x > 4 days THEN GOTO unit_x
```

If the hypothesis is wrong, over the years that the CAI program is in use, many students may experience the boredom of a lesson they do not need or the frustration of missing a lesson they do need. This situation can arise, not only when an incorrect hypotheses is entered originally, but also when shifts in the student population render an originally valid hypothesis inappropriate.

Unlike conditional branching in conventional CAI, the decision process described here gives students a test on the indicated objective. The test is necessary to provide a basis for ILP to improve or confirm the current set of classification rules. If the test is failed, confidence in the current set of rules is increased and the student enters a unit covering the objective. If the test is passed, ILP tries to modify the current rules to accommodate the disconfirming evidence and the student is tested on one of the other prerequisites implicated by the learning hierarchy.

This chapter focuses on the design of ILP, and shows how it relates to previous inductive learning programs. The description of ILP is preceded by a section reviewing some of the more prominent and relevant inductive learning research.

4.1 Inductive learning

When organisms learn to behave appropriately to a class of objects in their environment as a result of experience with objects forming a proper subset of that class, they are said to have learned 'by induction' or 'from examples'. This kind of learning has long been viewed by philosophers and psychologists as central to forms of intelligent behavior ranging from the formulation of scientific theories to the adaption of animals to changing environments.

In the field of AI, conjecture by Von Neuman and others concerning the complexity of designing a complete intelligence relative to that of designing an initial kernel which incrementally approaches intelligence through learning, has resulted in a search for a theory and methodology of machine learning. The study of machine learning, which includes inductive learning, learning by analogy, and 'learning by being told', is now a burgeoning sub-field of AI (Dietterich, 1982; Michalski, Carbonell & Mitchell, 1983, 1986). In their comprehensive bibliography, Utgoff and Nudel (1983) listed 572 publications dealing with machine learning. A more recent machine learning bibliography (Kedar-Cabelli & Mahadevan, 1986) lists over 300 books and articles published between 1980 and 1984, 86 of them concerned specifically with inductive learning. Although a strong theory of machine learning has not yet emerged, certain principles of inductive learning have been successfully applied to problems as diverse as diagnosis of plant disease, symbolic integration, scientific theorizing, and language acquisition.

The recent growth of interest in machine learning is partly a reaction to the expert system paradigm which dominated AI in the 1970's. While acknowledging the success of many expert systems, Langley (1987, p. 99) points out that their expertise is limited to highly specific domains, and that research into the mechanisms of learning provides "hope that our final theory of intelligence will consist of more than a few basic search techniques, along with the statement that domain-specific knowledge can be used to direct one's search".

Work in machine learning is also motivated by practical considerations. Michalski (1983) mentions two types of applications which he hopes will result from work such as his on the more fundamental aspects of inductive learning:

- Building knowledge bases for expert systems. The conventional method of developing expert systems (that is, by encoding rules or information obtained by interrogating human experts), is a very tedious process which places a rather severe limit on the size of knowledge bases so obtained. Michalski would break the knowledge acquisition "bottleneck" through the use of inductive learning.
- Exploratory data analysis in the experimental sciences. Inductive learning methods may extend the automation of scientific work past that presently available with

statistical techniques such as regression and factor analysis. Michalski's hopes for this area of application rest on the ability of inductive learning methods to uncover hidden logical structure in experimental data.

Classification rules developed by an inductive learning system can be more accurate than those obtained directly from human experts. Michalski and Chilausky (1980) reported an experiment in which their inductive learning system AQ11 was applied to the problem of discovering diagnostic rules for 15 soybean plant diseases. The goal of the project was to compare the performance of rules obtained through consultation with a plant pathologist with the performance of rules generated by AQ11. AQ11 was supplied with descriptions of 290 diseased soybean plants, along with an expert's diagnosis of each plant. Here is one diagnostic rule learned by AQ11 for the disease Bacterial pustule:

(leafspots=present & leafspot_size=large & stem=normal & roots=rotted) OR
(time=may & precipitation=normal & leaves=abnormal & leafspots=present.yellow)
⇒ bacterial_pustule

The expert-generated rules and the machine-generated rules were both tested on an additional 340 examples of diseased plants. For each plant, the rules gave a list of alternative diagnoses. The machine-generated rules put the correct diagnosis at the top of the list in 97.6% of the test cases, in contrast to only 71.8% for the expert-generated rules.

4.1.1 Inductive learning by parameter adjustment

The earliest machine learning systems can be characterized as using classified examples to obtain polynomial discriminant functions (Nilsson, 1965). In psychological terms, the weights or coefficients learned by such a system represent strengths of association between features describing the examples and example classifications. Learning by parameter adjustment has played a major role in neural modeling, cybernetics, pattern recognition, control theory, statistical decision theory, and related studies which arose with advent of electronic computers in the post-war period.

The study of artificial intelligence has its roots in the neural models, or self-organizing systems, pioneered by McCulloch and Pitts (1943). The neural modeling, or connectionist, approach emphasized parallelism and the simulation and application of biological mechanisms for adaption. Interest in the connectionist approach waned in the AI community partly as a result of Minsky and Papert's (1969) analysis showing certain fundamental limitations in the abilities of Rosenblatt's (1958) perceptron, the most influential of the early neural modeling systems. Perhaps because of the availability of much more powerful parallel computing hardware, interest in connectionism has recently been rekindled (Rumelhart & McClelland, 1986).

The inductive learning systems reviewed in the following sections learn concepts expressed in a logical or quasi-logical language, or semantic nets which are easily translated into such a language. According to Carbonell et al. (1983) these systems differ from the earlier parameter adjustment systems in that they can learn "symbolic representations" expressing "higher level knowledge". Whether or not one accepts such claims, it does seem that authors would be better able to express their hypotheses as logical statements than as neural networks.

4.1.2 Inductive inference and description spaces

The process by which human teachers adapt their tutoring skills to particular instructional settings can be viewed as a kind of inductive inference or learning. In this case, the examples are events involving the success or failure of various instructional treatments applied to differing types of students. Although in this section (and in all subsequent sections prior to section 4.2) the problem of constructing computer programs capable of inductive learning is discussed in general terms, the reader may find it useful to consider how each of the points covered relates to instructional applications.

Deduction and induction are complementary and reciprocal forms of logical inference which philosophers have studied and argued about for centuries. Deduction involves reasoning from general assertions to specific conclusions via established rules of inference such as hypothetical syllogism, modus ponens, and so on. Deductive inference is useful because it is truth-preserving: if the premises are true then the conclusion must be true as well. However, deduction is sometimes said to be limited by the constraint that what the conclusion states is always implicitly contained in the premises, and that therefore deduction cannot generate new knowledge.

Induction, on the other hand, involves 'going beyond the information given'. The rules of inductive inference are precisely the reverse of those for deductive inference. For example, consider the simplification rule (Kalish, Montague & Mar, 1980, p. 60) from deductive logic:

$$(\phi \text{ AND } \psi) \text{ THEREFORE } \phi$$

To use this as a rule for inductive inference, ϕ is taken as the premise and (ϕ AND ψ) as the conclusion. If ϕ is false, then (ϕ AND ψ) will also be false. For example, if ordered_pizza is false then (ordered_pizza AND no_bugs_in_program) is false regardless of the truth of no_bugs_in_program. It is the falsehood-preserving quality of inductive inference which is behind Popper's (1959) observation that scientific theories can be falsified by experiment, but not proved.

Simon and Lea (1974) cast inductive learning as search for a true hypothesis through a hypothesis space (or description space) guided by examples. Inductive inference can be viewed as simply a strategy for restricting the search through the hypothesis space by only considering those hypotheses which agree with (i.e., deductively infer) the known examples. Of course, the primary limitation of this principle is that there may be many false hypotheses which agree with the known examples. The interpretation of inductive learning as search unifies the problem with other areas of AI and, to some extent, allows their methods to be brought to bear.

The language used to specify hypotheses or concept descriptions is important because it defines the size and structure of the description space. A highly expressive language defines a large and perhaps infinite description space. A very restricted language may define a small description space which can be exhaustively searched, but it may not be capable of expressing the range of possible concepts demanded by the application. Ideally, one wants to use a language which can express all and only those concepts necessary to the application.

The problem of deciding when an example matches a concept description can often be simplified by specifying examples in a language which is a subset of the description language[17]. This technique, known as the **single representation trick** (Dietterich, 1982), has been widely used in machine learning programs.

Mitchell (1982) provided a set theoretic interpretation of the structure of the description space based on the idea that when a description D1 defines a set of examples which is a proper superset of the set of examples defined by description D2, then D1 is more-general-than D2. The more-general-than relation imposes a partial ordering on the description space. Mitchell pointed out that "this partial ordering provides a powerful basis for organizing the search through the [description] space".

To make these ideas more concrete, let's consider a very simple description language capable of describing classes of objects having the attributes of size (BIG or SMALL), color (RED, BLUE, or GREEN), and shape (ROUND or SQUARE). In this language, concepts (i.e., classes of objects) are properly described by a conjunction of **attribute-value pairs** (e.g., size=BIG & color=GREEN), but it happens that we can drop the attribute name without introducing ambiguity (BIG & GREEN). Figure 4.1 represents a portion of the description space arranged in a hierarchy with the most general concept at the top (level 0) and the most specific concepts at the bottom (level 3). A picture of the entire description space would show 12 concept descriptions at level 3, and 16 concept descriptions at level 2. A line connecting two concepts indicates that the higher concept is more-general-than the lower concept. The most general concept covers, or matches, all examples, and each of the most specific concepts covers a single distinct example.

---

[17] This means that the set of all expressible examples will be a subset of the set of all expressible descriptions.

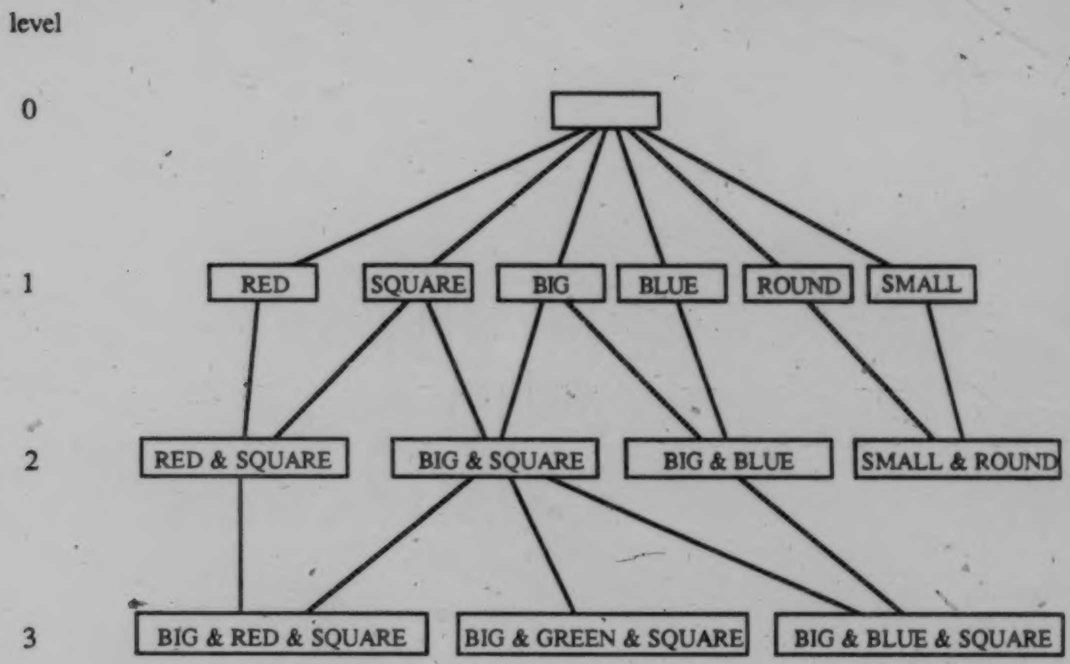

level
0
1
RED
SQUARE
BIG
BLUE
ROUND
SMALL
2
RED & SQUARE
BIG & SQUARE
BIG & BLUE
SMALL & ROUND
3
BIG & RED & SQUARE
BIG & GREEN & SQUARE
BIG & BLUE & SQUARE

Figure 4.1 Part of a description space partially ordered by the more-general-than relation.

4.1.3 Learning conjunctive concepts

In their seminal psychological study of concept learning, Bruner et al. (1956) asked subjects to form concepts from positive and negative examples easily represented by attribute-value pairs. They found that one of the most frequently used strategies worked as follows:

1) initialize the description of the concept to equal the first positive example encountered
2) every time a new positive example is encountered compare it to the current description and delete from the current description any attribute-value pairs which conflict with the new example.

This strategy finds the **most specific generalization** of the known positive examples. It can be applied quite successfully to problems like that represented in Figure 4.1. For instance, if the current description is BIG & GREEN & SQUARE and one encounters the example BIG & RED & SQUARE, then the new description is BIG & SQUARE.

Like the other learning strategies considered in this section, this simple generalization procedure is incapable of learning disjunctive concepts such as GREEN & (BIG OR SQUARE). Bruner et al. found that, in fact, humans have considerable difficulty learning disjunctive concepts from examples. Another significant shortcoming to the strategy is that it does not take advantage of information provided by negative examples.

Winston (1975) extended this basic generalization strategy to the learning of structural descriptions. Structural descriptions can represent concepts consisting of inter-related components and can be distinguished from attribute descriptions, like those depicted in Figure 4.1, which represent only global properties (Dietterich & Michalski, 1983)[18]. Winston's program, ARCH, used semantic nets to describe concepts from a blocks world. When a new positive example was encountered, the first step was to find a mapping between components in the current description and components in the new example, then the description was generalized to cover the example. Sometimes negative examples were encountered which matched the current description, indicating that the mapping used in a previous generalization had been inappropriate. In this case the program backtracked to a previous generalization step and chose an alternate mapping. Thus, ARCH can be characterized as performing a depth-first search of the description space, a search made necessary because the description language permitted more than one most specific generalization. To forestall combinatorial explosion in the number of descriptions considered, ARCH required a rather sympathetic source of examples (Dietterich & Michalski, 1983).

---

[18] Dietterich and Michalski noted that, in principle, one can represent structural descriptions by attribute value pairs, but this tends to result in an unwieldy explosion in the number of attributes.

SPROUTER (Hayes-Roth & McDermott, 1976) learned conjunctive structural descriptions similar to those learned by Winston's program. But SPROUTER used a **beam search** procedure to consider several most specific generalizations simultaneously. Beam search, illustrated in Figure 4.2, is a non-exhaustive search which offers a compromise between depth-first and breadth-first search in problems where a heuristic utility function is available for ordering the descriptions under consideration. When a positive example is encountered: (1) all generalizations needed to cover the example (including descriptions already matching the example) are put into a list, (2) members of the list matching any known negative examples are deleted, (3) the list is ordered according to the utility function, and (4) the list is truncated to some number of descriptions (the maximum beam width) fixed by the user. The list becomes the new set of current descriptions. When a negative example is encountered, it is compared to all current descriptions, and any descriptions which match it are deleted from the set. The use of heuristic directed beam search made SPROUTER more robust with respect to the order and quality of training examples than ARCH. Also, the elimination of backtracking meant that SPROUTER did not need to store previously encountered positive examples, although it still had to store all negative examples.

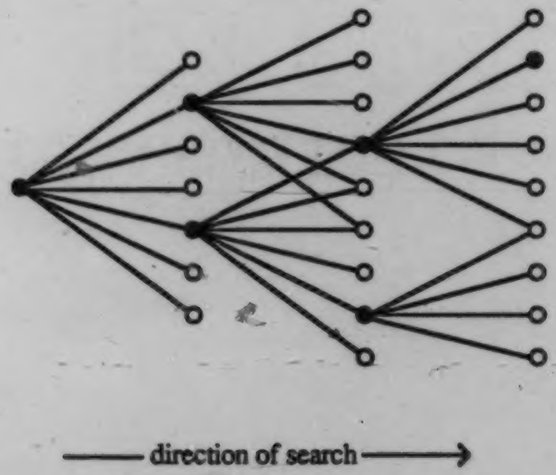


Figure 4.2 A beam search with maximum beam width = 2.

Mitchell's version space approach to inductive learning (Mitchell, 1977; 1982; Mitchell, Utgoff, & Banerji, 1983) can be viewed as a logical culmination of the methods reviewed in this section, and for this reason has been widely cited and discussed in the machine learning literature. In a sentence, what separates the version space approach from previous methods is the realization that, in general, there is no reason to focus primarily on positive examples, and that certain advantages result from treating positive and negative examples in a symmetric and complementary fashion.

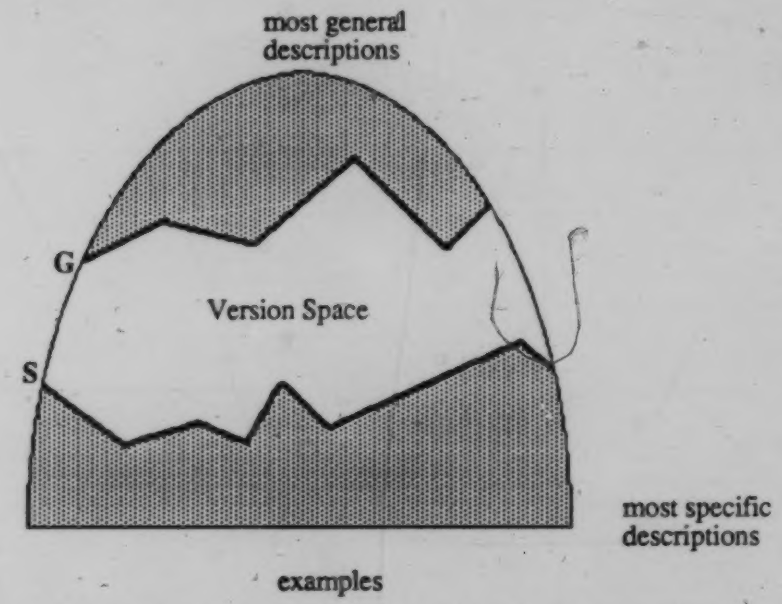


Figure 4.3 The boundary sets S and G are used to compactly define the version space.

In Mitchell's terminology, the version space is the set of descriptions which have not yet been ruled out by the examples. Before any examples are encountered, the version space is equivalent to the entire description space. Examples have the effect of eliminating descriptions from the version space, until (one hopes) there is only a single description left and the procedure can terminate to report with confidence that the concept has been learned. Of course, in most conceivable applications, the description space is far too large to be stored explicitly and exhaustively searched. As shown in Figure 4.3, Mitchell exploits the partial ordering of the description space by compactly representing the version space with two boundary sets: S, the set of most-specific-generalizations; and G, the set of most-general-specializations.

A **candidate elimination algorithm** updates the boundary sets when a new example is encountered. A positive example e^+ will (1) cause descriptions in S which do not cover e^+ to be replaced by descriptions generalized just enough to cover e^+ but constrained to be more specific than, or equivalent to, at least one description in G, and (2) cause descriptions in G not covering e^+ to be deleted. A negative example e^- will (1) cause descriptions in G which cover e^- to be replaced by descriptions specialized just enough to exclude e^- but constrained to be more general than, or equivalent to, at least one description in S, and (2) cause descriptions in S which cover e^- to be deleted. The learning process halts when S and G have converged to both contain the same, single description. The need for the deletion operations will depend on the structure of the description space.

Table 4.1 The candidate elimination algorithm learning a simple concept.

Example	Classification	Version space
1) BIG & GREEN & SQUARE	+	S = {BIG & GREEN & SQUARE} G = {}
2) SMALL & RED & SQUARE	-	S = {BIG & GREEN & SQUARE} G = {BIG, GREEN}
3) BIG & BLUE & SQUARE	+	S = {BIG & SQUARE} G = {BIG}
4) BIG & BLUE & ROUND	-	S = {BIG & SQUARE} G = {BIG & SQUARE}

Table 4.1 shows how the candidate elimination algorithm is guided by four examples to learn a concept from the description space represented in Figure 4.1. In simple description languages of this type, S never contains more than one member, and negative examples cannot cause deletion of elements from S. Notice that after the second example, there are two specializations (BLUE, and ROUND) which exclude the negative example but which

cannot become members of G because they are neither more general than, nor equivalent to, the description in S.

Two important advantages of the version space approach over previous methods for learning conjunctive concepts are that no examples need to be saved, and that it can identify when a true description of the concept has been obtained. The version space approach appears to provide optimal learning performance in description spaces of the kind illustrated in Figure 4.1, but is often impractical for more realistic applications. For instance, it may be that there are many different ways of expressing the same concept, and that the equivalence of any two expressions is generally undecidable. In such cases the program may be unable to determine when it has finished learning. The version space approach will not work with languages permitting unrestricted negation because, in the case of such a language, G will always only contain a single description consisting of the conjunction of the negation of all negative examples. Also, for many applications, S and G are simply too large to be maintained and searched. In these situations one must prune the search, perhaps as in SPROUTER, and thus to some extent relinquish the advantages of the version space approach.

4.1.4 Learning disjunctive concepts

Some applications of inductive learning require a description language permitting both disjunction and conjunction. But for both humans and machines, admitting disjunction can make the learning task much more difficult. One reason is that the most specific generalization will simply be the disjunction of all positive examples encountered, known as the **trivial disjunction**. Thus, the standard version space approach cannot be applied because S and G will not converge. The whole purpose of inductive learning is to make inferences about examples which have not been sampled, and unless trivial disjunction is avoided, inductive learning is reduced to merely rote learning.

One way of simplifying the problem is to restrict the learned descriptions to a disjunctive normal form (DNF). DNF expressions are disjunctions of conjunctive clauses, such as (with $\vee$ as the symbol for disjunction):

$$(\text{GREEN} \,\&\, \text{BIG}) \vee (\text{GREEN} \,\&\, \text{SQUARE})$$

This is not necessarily a serious restriction because for every expression in standard logical form there exists an equivalent expression in DNF. One can imagine a module in the system which tries to convert learned DNF expressions into more compact and human-readable standard logic expressions for output to the user.

Three goals or desired properties can be set for DNF descriptions learned from positive and negative examples:

- consistency, which is satisfied by the exclusion of all negative examples
- completeness, which is satisfied by the inclusion of all positive examples
- parsimony, which is satisfied by minimizing the number of conjunctive clauses

The trivial disjunction is consistent and complete but, assuming a better solution exists, completely without parsimony.

4.1.4.1 Disjunctive version spaces

Mitchell suggested a modification of the version space approach to allow learning of DNF descriptions. The procedure assumes that the set of positive examples (POS) and the set of negative examples (NEG) are known a priori. It follows a strategy, referred to here as the positive example deletion strategy (Table 4.2), which treats each conjunctive clause as a separately learned sub-description.

When using this strategy with the version spaces approach, in step 3 the most specific set S is set to contain e^+, and the most general set G is initialized with the most general clause. The candidate elimination algorithm is applied with every example in NEG. This has no effect on S but will develop G.The final DNF description learned will depend on the order of the positive examples selected in step 2 and the clause selected in step 4. There is no guarantee that the learned description will be the most parsimonious.

Table 4.2 The positive example deletion strategy.

1) Initialize the description to have zero clauses.

2) Select an example e^+ from POS.

3) Develop a set G of clauses which cover e^+ and exclude all members of NEG.

4) Select a clause D from G.

5) Append D to the description by disjunction.

6) Delete from POS all examples covered by D.

7) If POS is empty then halt, else go to step 2.

The requirement of prior knowledge of examples can be a nuisance when the task is to incrementally learn from a stream of positive and negative examples. Unless all negative examples are available for every iteration of step 3, a clause could be learned which is not consistent. This means that to maintain consistency, the learning process must start from scratch every time a new negative example is encountered.

Mitchell et al. (1983) seem to have abandoned the consistency requirement in the design of LEX, a program which learns heuristics for solving symbolic integration problems from a stream of examples. LEX starts with a single specific set S_1, and allows positive examples to generalize S_1 and negative examples to specialize G, until a positive example is encountered which cannot be accommodated by a generalization of S_1. At this point an additional specific set S_2 is created and initialized with the new example. Each subsequent positive example is either assigned to one of the existing specific sets (possibly serving to generalize it), or it starts its own specific set if generalization of one of the existing specific sets is not possible. The description space of LEX is such that each specific set contains only one sub-description, so the learned DNF description is simply the disjunction of the members of all specific sets.

Mitchell et al. did not discuss the troublesome case of a new negative example which matches one of the specific sets. Consistency is sacrificed if the example is ignored, but completeness may be lost if the specific set is somehow specialized to exclude it. Bundy et al. (1985) list several flaws in LEX, and point out that they stem from LEX's inability to properly assign positive examples to clauses:

> The new shell [clause] gets preferential treatment when it comes to allocating the new positive instances [examples] between shells, whereas the old shell gets preferential treatment when it comes to allocating the old positive instances. There is no reason to assume that this is the correct division of positive instances. Negative instances, of course, apply to both shells. The ad hoc nature of the division of the positive examples makes it very unlikely that LEX will learn the correct disjunctive rule. . . . We know of no way of ensuring that the division of instances is performed correctly the first time so that the program never needs to backtrack and reassign the instances. Storage of all training instances is required to enable this backtracking. (p. 168-171)

4.1.4.2 Michalski's inductive learning methodology

Michalski has developed a very extensive methodology for inductive learning (Michalski, 1983), which has strongly influenced the design of the inductive learning procedure presented later in this chapter. The summary given here attempts to cover only the most fundamental and relevant aspects of his methodology, and necessarily provides a simplified view.

In the terminology of inductive learning, bias is any aspect of the learning system, aside from the examples, which influences the selection of concept descriptions (Utgoff, 1986). Bias is important because it is often the case that the examples are not sufficient for concept selection: either many descriptions exist which are complete and consistent, or no descriptions exist which are complete and consistent. Bias can be introduced in the form of restrictions inherent in the description language and description space, search heuristics, explicit concept selection criteria and so on.

Two principles seem to serve as the foundation of Michalski's work:

1) There is a need for a general inductive learning system that is applicable to many different domains. General applicability demands that this system work with an expressive and relatively unrestricted description language, with provision for domain-specific bias introduced by the user as background knowledge.

2) Inductive learning systems should have a "bias toward comprehensibility", that is, they should tend to learn concepts which are understandable to humans, and can be readily translated into natural language. For instance, sentences in the description language should contain only a few conjunctions and disjunctions, and no more than one level of parentheses. This bias emphasizes not only that humans are often the consumers of learned concepts, but also that they can be a very useful source of feedback and domain-specific knowledge throughout the learning process.

Michalski designed a description language to facilitate inductive learning called **annotated predicate calculus** (APC). APC is in several respects more expressive than standard first order predicate calculus. For instance, in addition to the standard universal and existential quantification, it permits a numerical quantifier. Instead of having to describe a biped as:

$$\exists x\ \exists y\ [\mathrm{leg}(x)\ \&\ \mathrm{leg}(y)\ \&\ x \neq y]$$

one can use a more natural expression something like:

$$\exists(2)x\ [\mathrm{leg}(x)]\ .$$

APC expressions are composed of quantifiers, logical connectives ($\&, \vee, \sim, \Rightarrow, \Leftrightarrow$), and descriptors (predicates, variables, and functions). Descriptors are either defined by the user, or are built from user-defined descriptors by a process of **constructive generalization**. With each descriptor they define, users also supply an annotation, which indicates auxiliary background information about the descriptor such as its domain and type. In standard predicate logic, much of this information would have to be represented as additional axioms.

Michalski's methodology recognizes three types of descriptors. Simply defined these are: nominal descriptors, which have an unordered value set; linear descriptors, which have a totally ordered value set; and structured descriptors, which have value sets whose members are arranged in a generalization tree.

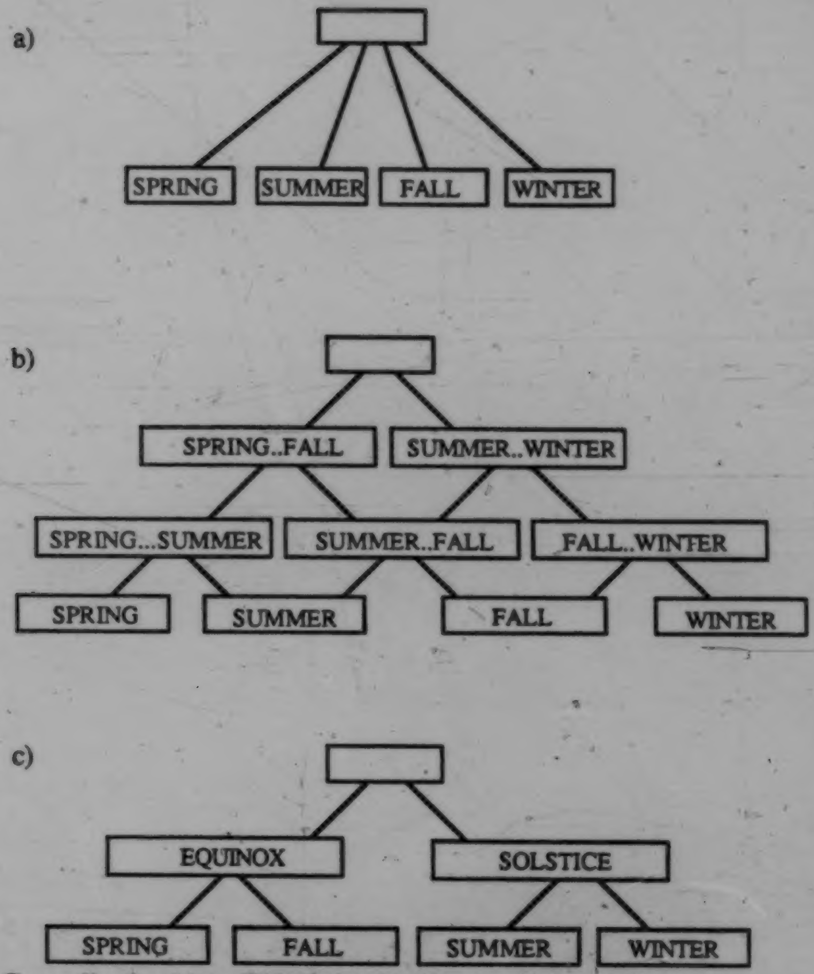

a)
SPRING
SUMMER
FALL
WINTER
b)
SPRING..FALL
SUMMER..WINTER
SPRING...SUMMER
SUMMER..FALL
FALL..WINTER
SPRING
SUMMER
FALL
WINTER
c)
EQUINOX
SOLSTICE
SPRING
FALL
SUMMER
WINTER

Figure 4.4 Generalization hierarchies determined by specifying attribute type.

Like other kinds of annotation, descriptor typing is just a way of allowing the user to supply valuable background knowledge which will bias the learning process. It can be viewed as a way of specifying a generalization hierarchy over the value set for the descriptor. Consider the variable SEASON with the value set {SPRING, SUMMER, FALL, WINTER}. When specified as a nominal variable it has an implicit flat generalization hierarchy as shown in Figure 4.4a. When specified as an ordered variable it has the implicit generalization hierarchy shown in 4.4b. If neither of these capture the user's intuitions about how the values should be generalized, the hierarchy can be given explicitly as structured variable as shown in Figure 4.4c. Because the structured type may also include linear nodes (not illustrated here), it is really a combination of the nominal and linear types.

In order to facilitate inductive learning, both APC and its predecessor VL_1 (Michalski, 1973; Michalski & Chilausky, 1980) are fundamentally disjunctive normal forms. However, they do allow restricted "internal disjunction", which can be particularly useful in expressing ranges on linear descriptors. For instance, a VL_1 rule learned by AQ11 for diagnosing the soybean plant disease anthracnose contained the term [time = Aug..Oct], indicating a period in the growing season from August to October inclusive.

Michalski views inductive learning as state-space search where the initial state is the collection of examples, and the goal state is an APC assertion that implies the examples (i.e., is consistent and complete), satisfies user-supplied background information, and satisfies a user-supplied preference criterion. There are three types of rules which can be applied in conducting the search: generalization rules transform a description into a more general description, reformulation rules transform a description into a logically equivalent description, and specialization rules transform a description into a less general description. Although reformulation and specialization rules (the truth preserving rules of deductive logic) are useful in inductive learning, it is the generalization rules which are of primary importance.

A few examples are given here of generalization rules used in Michalski's methodology. The symbol ::> is used for an implication linking a concept description with a concept name. The symbol $\mathrel{|\!<}$ is interpreted as "generalizes to". A and B indicate arbitrary APC expressions. K indicates a predicate asserting the name of a concept.

- The dropping condition rule:

$$A \,\&\, B ::> K \mathrel{|\!<} A ::> K$$

- The adding alternative rule:

$$A ::> K \mathrel{|\!<} A \vee B ::> K$$

- The closing interval rule:

$$(A \;\&\; L{=}c ::> K) \;\&\; (A \;\&\; L{=}d ::> K) \;\vert\!< \; (A \;\&\; L{=}c..d ::> K)$$

where L is a linear descriptor, and c and d are values of descriptor L such that $c < d$. The term L=c..d means "L has a value no less than c and no greater than d".

- The climbing generalization tree rule:

$$(A \;\&\; S{=}c ::> K) \;\&\; (A \;\&\; S{=}d ::> K) \;\vert\!< \; (A \;\&\; S{=}f ::> K)$$

where S is a structured descriptor, and f is the most specific node having nodes c and d as descendants.

There are also rules for constructive generalization which generate new descriptors and thereby transform the description space. For instance, the generating chain properties rule is applied to transitive relations to construct descriptors identifying the extrema of the relation. In a blocks world with a user supplied descriptor ABOVE(x,y), this rule could be applied to construct the descriptors MOST_ABOVE(x), meaning "top", and LEAST_ABOVE(x), meaning "bottom". Previous work with constructive generalization (e.g., Lenat, 1983, p. 252) has shown the value of this "find extrema" heuristic.

Michalski's methodology requires that the user specify preference criteria for evaluating the many descriptions that may be generated by applications of the specialization, reformulation, and generalization rules. The user selects criteria from a menu which may include consistency, completeness, ease of comprehension, cost of evaluating descriptors, and so on. The selected criteria are combined in a sequence of criterion-tolerance pairs:

(c_1, t_1) (c_2, t_2) ...

The tolerances are expressed as percentages of the criterion measure. When a number of descriptions are under consideration, they are first evaluated on criterion c_1, and the best description or those which match within tolerance t_1 are retained. Retained descriptions are evaluated on c_2 and so on, until only one description is retained or the sequence of criteria is exhausted.

The inductive learning algorithm proposed by Michalski focuses on the generation of a set of descriptions called a **star**. A star, denoted $G(e^+|NEG)$ where e^+ is a positive example and NEG is the set of negative examples, is the set of all maximally general conjunctive clauses in APC matching e^+ but not matching any members of NEG. In practice it is necessary to aim for a bounded star $G(e^+|NEG,m)$ containing only the m most preferable descriptions in $G(e^+|NEG)$.

Michalski's learning algorithm is quite similar to Mitchell's disjunctive version spaces approach in that it also uses the positive example deletion strategy outlined in Table 4.2. In Michalski's methodology the bounded star $G(e^+|NEG,m)$ substitutes for the set G in step 3. In step 4, D is selected according to the preference criterion. Also, reformulation rules are applied to the final description in an attempt to derive a contracted expression.

The major difference is that, while Mitchell's candidate elimination algorithm is only practical for very restricted description languages, Michalski gives an efficient star generation procedure based on beam search which works within APC, an expressive all-purpose language. The star generation procedure is summarized as follows:

1) Create a set PS (partial star) whose members are all single-descriptor generalizations of e^+. For instance, if e^+ is BIG & GREEN & SQUARE, then PS = {BIG, GREEN, SQUARE}. Constructive generalization rules may be applied at this point to introduce other single-descriptor generalizations. Throughout this and subsequent steps, PS is ordered according to the preference criteria and truncated to contain the best m descriptions.
2) Descriptions in PS are tested for consistency and completeness. All consistent and complete descriptions are taken out of PS and put in a SOLUTION list. All consistent but incomplete descriptions are taken out of PS and put in a CONSISTENT list. If the size of the CONSISTENT list exceeds a user-supplied parameter, or the cpu time exceeds some bound, then go to step 4.
3) Apply specialization rules to the descriptions in PS in an attempt to make them consistent. Go to step 2.

4) Apply generalization rules to the descriptions in CONSISTENT to make them more complete.

5) The final bounded star $G(e^{+}|NEG,m)$ is apparently a union of the SOLUTION list with the best descriptions from the CONSISTENT list.

4.1.4.3 Learning decision trees

Hunt et al. (1966) developed a concept learning system (CLS) which has more recently been refined and applied to chess end game problems by Quinlan (1983). The task addressed by CLS, and Quinlan's ID3, is to learn classification rules from examples expressed as attribute vectors, and to use those rules to classify new examples. It is assumed that the attributes are nominal variables with rather small value sets.

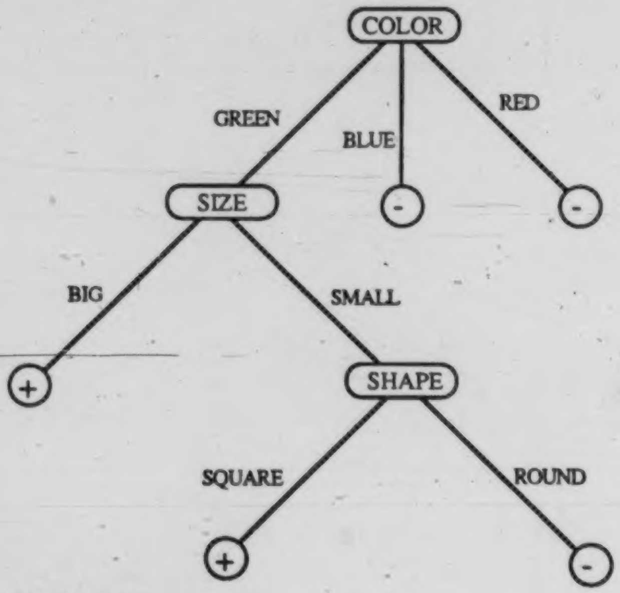


Figure 4.5 A CLS/ID3 decision tree.

What really distinguishes CLS and ID3 from other machine learning systems is that the classification rules they learn are expressed as decision trees. Every inner node of a decision tree represents a test of one of the attributes. There is a child node for each possible outcome of the test (i.e., each member of the value set of the tested attribute). In classifying a new example, the system starts with the test at the root node and branches down the tree applying tests in sequence until a leaf node is reached. The leaf nodes indicate the classification decision based on the series of test outcomes. For instance, the concept GREEN & (BIG v SQUARE) is represented by the decision tree shown in Figure 4.5.

There is a simple and efficient procedure which will generate a complete and consistent decision tree from a set C of classified examples, a list A of attributes, and a root node n:

```
build_tree(C,n,A);
1) if C is empty then arbitrarily label node n as + or - and return;
2) if C contains all positive examples then label node n as + and return;
3) if C contains all negative examples then label node n as - and return;
4) select an attribute $\alpha$ from A and partition C into disjoint sets $C_1$, $C_2$, ..., where
   $C_i$ contains those members of C having the ith value of $\alpha$;
5) delete $\alpha$ from A;
6) for each value of $\alpha$:
       create a new node $n_i$ which is a child of n;
       call build_tree($C_i$,$n_i$,A);
```

The structure and number of nodes in the resulting decision tree will be determined by the order of attributes selected in step 4.

The goal of inductive learning is not only a consistent and complete concept description, but also one which can correctly classify new examples. Unless some heuristic is used to guide the ordering of attributes, the procedure described above may generate a decision tree equivalent of the trivial disjunction, that is, a tree in which each leaf covers only one or zero examples. To avoid such a tree, and to aim for a tree which is in some sense minimal, CLS and ID3 apply heuristics that try to select the attribute best separating the positive and negative examples.

Quinlan, taking an information-theoretic approach, viewed a decision tree as an information source which, given an example, conveys a message about the classification of the example. From information theory the average information content, or entropy, of the messages is:

$$-q^{+}\log_2 q^{+} - q^{-} \log_2 q^{-}$$

where q^{+} and q^{-} are the proportions of positive and negative examples in C.

All decision trees created from the same set of examples will have the same entropy. Also, the entropy of the decision can be regarded as being distributed over the nodes of the decision tree, with larger trees conveying less information per node. Therefore, in order to generate parsimonious trees with few nodes, Quinlan adopted a steepest ascent strategy which selected the attribute accounting for the greatest portion of the remaining entropy.

Another difference between CLS and ID3 is that Quinlan has developed means of drawing samples from C which enable ID3 to deal efficiently with large numbers of examples. Quinlan found that, in a task involving 715 examples and 49 attributes, ID3 learned a decision tree containing 177 nodes in 34 seconds on a Cyber 72.

ID3 is one of the most efficient programs yet devised for learning disjunctive concepts, but this advantage has been gained at the expense of significant limitations in the description language. The most important of these are the restriction in the size of the attribute value sets, and the inability of the procedure to generalize over, and represent, ranges within the value sets of ordered attributes (as Michalski's methodology does with linear descriptors).

4.1.5 Learning from noisy examples

All of the inductive learning systems described to this point have assumed that the examples used for learning are free from errors or noise, and that a consistent and complete description exists somewhere in the description space. But in many potential applications of inductive learning, the information comprising the examples is sometimes incorrect, or is just not sufficient to support a discriminating description. For instance, the critic module which produced examples in LEX (Mitchell et al., 1983) occasionally gave the wrong classification for an example. It is not clear what effect this had on the performance of LEX's learning module, which had no special mechanism for dealing with noisy examples.

Quinlan (1986) investigated various extensions of ID3 which support learning from noisy examples. The principle is that, in step 4, instead of always choosing the best attribute to partition C, ID3 may decide that the hetrogeneity of C (mixed positive and negative examples) is due to noise only and declare the current node to be a noisy leaf. The noisy leaf is labeled with the classification (+ or -) held by the majority of the examples in C.

The difficulty comes in deciding when there are no attributes capable of providing a good enough partitioning of C. One strategy is to declare the node to be a noisy leaf when the entropy accounted for by the best available attribute is less than some selected threshold. In applying this strategy, Quinlan found that any threshold setting "sufficiently large to prevent testing irrelevant attributes significantly degraded the performance of the procedure when the attributes were indeed adequate". He found that a much better approach was to reject the use of an attribute for partitioning C when the attribute values held by the examples in C were statistically independent of the classifications of those examples. Independence was measured by a chi-square test with a confidence level of 99 percent.

Lee and Ray (1986a) developed a probabilistic rule generator (PRG) that learns a modified form of Michalski's VL1 descriptions. PRG seems to be an effective merger of the methods used by Michalski in AQ11 and Quinlan in ID3. Because it learns disjunctive concepts, tolerates noisy examples, and recognizes both linear and nominal attributes, PRG comes closer than previous procedures to satisfying the requirements of the instructional application considered here.

The description language is a disjunctive normal form in which every conjunctive clause is weighted with an integer indicating the number of positive examples it covers. Clauses are composed of attribute-value or attribute-range pairs. For instance, the following description has three clauses with weights of 10, 3, and 5:

$$10[a_2=1..2] \vee 3[a_2=4] \vee 5[a_1=0..1][a_2=2..4]$$

Like most of the programs which learn disjunctive concepts, PRG forms each clause separately by following the positive example deletion strategy given in Table 4.2. But unlike previous programs using this strategy, PRG does not base clause formation on a single positive example selected as a seed. Instead, it does a general to specific breadth-first search guided by an entropy reduction heuristic, followed by an attempt to expand the resulting clause to cover regions of the example space not occupied by negative examples. An important advantage of this approach is that it does not risk basing an entire clause on what may be an erroneous example, and is more considerate of global properties of the example set.

Lets consider how the top-down search works with linear attributes. Like ID3, PRG starts with the most general clause covering the entire example space serving as the root node. For each attribute PRG obtains a boundary which best separates, according to the entropy criterion, positive and negative examples projected on to the attribute. The attributes are ranked by the entropy criterion and the best n are used to form up to n new clauses where n is the user supplied maximum search breadth[19]. The new clauses are specializations of the parent clause and each covers a sub-region of the example space covered by the parent. The sub-region can be viewed as that side of a plane partitioning the parent's example space which contains the greater number of positive examples. The process is repeated using each of the new clauses as a parent node to spawn further specializations.

Nominal attributes are temporarily "linearized" at each node so that this method can be feasibly applied. This means that an ordering is imposed on the value set of a nominal attribute according to the proportion of positive examples projected on to each value.

---

[19] Actually, the user supplies a separate search breadth for each depth of the search.

Lee and Ray (1986b, p. 442) recognized the importance of being able to bias rule learning with descriptions hypothesized by the user:

> There are many situations where we would like to construct a knowledge base initially as a set of hypotheses [clauses], introduced by a human expert, which are later systematically modified by experimental training data events. Indeed, one might say that this ordering of the learning process is analogous to the theoretical study of a problem's solution methods followed by practical experience with the problem, wherein modification or adaption of the initial rules or hypotheses occur. Required modifications may range from small perturbations of the hypotheses through major or minor deletions and additions of new rules.

This rule refinement feature is also a goal of the present study.

However, because PRG is based on the positive example elimination strategy, it requires all examples to be available at the start of the learning process. If a new example is encountered learning must start from scratch, receiving no benefit from previous learning. The positive example elimination strategy is incompatible with the direct manipulation of existing rules, no matter whether they come from the user or previous learning. Lee and Ray circumvented this fundamental restriction by having the user express the weight of initial clauses in terms of number of examples, the idea being that a hypothesized clause with a weight of n is as valid as an empirically developed clause based on n examples. To "simulate" a clause entered by the user, the set of positive examples was initialized with n artificial examples uniformly distributed throughout the example space covered by the entered clause.

An advantage of this method is that it allows the user a means of expressing the degree of certainty with which he believes each of the clauses comprising the initial description. Heavily weighted clauses are more robust and less malleable than uncertain clauses.

But artificial examples derived from a description do not bias the learning process in the same way that the description itself can. We know that there are often many descriptions that agree with any given set of examples. When we give artificial examples as a substitute for a description we are abandoning characteristics of the description which distinguish it from other descriptions which agree with those examples. These characteristics are just the kind of biasing information we hope the user will provide. Nevertheless, PRG should be regarded as a potential alternative to the inductive learning procedure presented in the next section.

4.2 An inductive learning procedure for adaptive instructional systems

The basic rationale for applying machine learning techniques to computer-based instruction has been succinctly stated by Michalski:

> Intelligent tutoring systems must be able to present material at a level of difficulty and detail suited to the state of knowledge of the student. In order to do so, the system must know and follow the student's changing knowledge. A desirable way of acquiring this knowledge is not by repeated direct testing but by learning from clues, behavior, and the implicit model of the student during tutorial sessions. Thus learning abilities are required not only from the student but from the tutor as well. (Michalski, 1986, p. 7).

One can imagine machine learning being incorporated in all of the three components of Hartley's framework for adaptive instruction (page 11). The expert model can acquire improved heuristics by analysing its own performance, as in LEX, or by recognizing when a student has obtained a better solution to a problem than the one it has provided, as in Kimball's self-improving tutor (Kimball, 1982). Advanced student models which incorporate learning allow for different instructional strategies to be tried on the model so the best can be selected for presentation to the student. Machine learning used in student models is constrained to be an emulation of human learning, and should model forgetting as well as acquisition (VanLehn, 1987).

A topic which has received little attention, and one with which the remainder of this chapter is concerned, is the use of inductive learning methods to improve diagnostic classification rules in the teacher model[20]. In this approach, student models are examples which an inductive learning procedure uses to refine heuristics initially entered by a course author. The hope is that the eventual performance of such a teacher model will parallel that of an experienced teacher who, upon encountering a student with some problem or other, applies diagnostic rules he has learned from experience with previous students. However, unlike the problem of incorporating learning in student models, there is no need to be constrained to reproducing the processes specific to human learning.

---

[20] Langley, Wogulis & Ohlsson (1987) applied inductive learning to the problem of reconstructing a few of the famous 110 bugs used in BUGGY (Burton, 1982) from primitive subtraction operators. A drawback to this technique is that the author must be able to supply a correct set of primitive operators.

ILP was designed and implemented for use in adaptive instructional systems of the kind based on learning hierarchy representations of the subject matter. It is assumed here that the classification rules learned by ILP are used to identify prerequisites that a student exhibiting difficulty with an objective (let us call it the **prime objective**) has forgotten. The outer control procedure, which invokes ILP, takes the following actions whenever such a student is encountered:

1) Recommendation Phase. One or more prerequisite objectives are recommended for testing by applying the learned classification rules to the model of the failing student.
2) Testing Phase. The student is tested on the recommended objective(s). If the result of a test is NON_MASTERY the student is routed to the appropriate unit for instruction. When all such units have been successfully completed the student is returned to the unit which was originally failed. If the result of the recommended test(s) is MASTERY, the corresponding objective(s) is removed from further consideration. If all recommended objectives are found to be in a state of MASTERY, then go to step 1 for a new recommendation.
3) Learning Phase. The student model and the test results are fed into ILP which uses the information to improve the classification rules.

There were three fundamental decisions made at an early stage of the project which strongly determined the design of ILP. First, an example is supplied to ILP only as a result of a student being explicitly tested on a prerequisite. Indirect evidence for an example is not allowed. For instance, when tests reveal mastery on k-1 out of k prerequisites, the student model is not automatically supplied to ILP as an example of a student in need of the remaining prerequisite. Although the use of such indirect evidence could speed the learning process, it was considered to rely too heavily on the assumption of absolute validity of the learning hierarchy.

Second, the form chosen for the examples was a simple conjunction or list of attribute-value pairs called an **attribute vector**. As seen in Chapter 2, attribute vectors are the closest we have to a standard form for student models in adaptive instructional systems. Even in highly domain specific intelligent tutoring systems, student models are rarely more complicated than lists of bug or skill names. In cases where the author expresses student models used within instructional units in a more powerful language, attributes judged to be relevant to the objective selection problem could be extracted from the more complicated form. The attribute vector will contain whatever information about the student the author chooses to include (for example, scores on aptitude tests, sex, age, objectives mastered, responses, and so on). The assumption is that this information will provide a better than chance basis for instructional control decisions.

Third, it was decided that classification rules hypothesized and directly entered by authors should be given high priority as a means of biasing the learning process. It was believed that this form of background knowledge offers the best starting point for the state-space search conducted by ILP.

The qualities which characterize this inductive learning task are:

- Single concept learning. It is possible that a student failing a unit has forgotten more than one of the prerequisite objectives. This means that hypotheses about different prerequisites are not in competition: a test confirming that one prerequisite has been forgotten does not disconfirm predictions that others have been as well. So, where there are k prerequisites to be considered, the problem can be broken down into k single concept learning subproblems.
- Incremental rule refinement. Students passing through the testing phase constitute a stream of examples for the learning phase. The learning procedure should try to improve the classification rules after each new example. If ILP waited until some

large set of examples had been collected, students represented by this set would not benefit from partially learned rules.

- Shifting of target concepts. Student populations may fluctuate over time necessitating changes in the classification rules. However means should be available to the author enabling him to (a) recognize when conditions are stable and (b) disable the testing and learning phases until he has reason to believe there is a change in the student population.
- Passive example acquisition. ILP has no control over the order and kind of students it encounters, so unlike some learning systems, it cannot accelerate concept discovery by conducting experiments (i.e., gathering examples) in critical regions of the example space.
- Disjunctive concepts. ILP must be capable of learning disjunctive concepts because there may be several reasons why a prerequisite is not mastered, and we should expect positive examples to cluster in disjoint regions of the example space. As Bundy et al. (1985) observed, in order to support learning of disjunctive concepts the learning system should keep all examples encountered.
- Noisy examples. The information used to construct and categorize the attribute vectors will be obtained from educational tests and from other means of observing students. Because these sources are subject to systematic and random error, the system must be able to tolerate situations where identical attribute vectors are categorized differently. In fact, even when there is no error, there is no guarantee that the set of attributes chosen by the author will provide a sufficient basis for discrimination between positive and negative examples.
- Linear or ordered attributes having large value sets. It seems clear that attributes cannot be restricted to the nominal type. Authors will want to use ordered attributes such as aptitude scores and response latencies. The need to support ordered attributes eliminates the CLS/ID3 approach from consideration for this application.

If some variation of the positive example deletion strategy were followed, it would seem necessary to (1) redo the entire learning process after each example and (2) introduce the author's hypothesized description only indirectly by initializing the example set with artificial examples derived from it (after Lee & Ray, 1986b). Instead, ILP starts the search at the description learned after the previous example and tries to improve it by applying generalization or specialization operations. Examples migrate between states of agreement and disagreement with the current rules, as the description is modified. But unlike LEX, ILP keeps track of the state of each example and uses disagreeing examples to guide the application of operations.

4.2.1 Classification rules and recommendation criteria

The classification rules learned by ILP can be viewed as having a production rule format:

<conjunctive_clause> <certainty_factor> ⇒ <prerequisite_name>

A certainty factor is associated with each clause. The certainty factor (CF) is an estimate of the probability that a student matching the clause really is in need of the prerequisite:

$$\mathrm{CF} = \frac{\mathrm{POSCOV}}{\mathrm{POSCOV} + \mathrm{NEGCOV}}$$

where POSCOV is the number of students matching (covered by) the clause who were found to not have mastery of the prerequisite, and NEGCOV is the number of students matching the clause who were found to have mastery of the prerequisite.

It is conceivable that the rules could be chained to perform a kind of deductive reasoning about the student, with certainty factors being combined according to one of the methods described by Spiegelhalter (1986) for dealing with uncertainty in expert systems. One such application would be a system which chained downward in the learning hierarchy to identify non-mastered prerequisites more than one level removed from the failed objective. However, this project deals only with immediate prerequisites, and omits consideration of rule chaining.

In the recommendation phase, the current student is compared to the left hand side (LHS) of all current rules. If none of the rules match, one prerequisite is randomly chosen as a recommendation[21]. If one or more rules match, the prerequisite indicated by the matching rule with the greatest CF is automatically recommended. Other matching rules will result in a recommendation only if they have CFs greater than a threshold supplied by the course author. This threshold will presumably reflect the author's subjective weighting of the cost of taking an extra pretest versus the cost of returning the student to the original objective with one prerequisite still not mastered.

4.2.2 Attributes

For each node in the learning hierarchy having more than one prerequisite, the author specifies a set of attributes which will comprise the attribute vectors used by ILP for building classification rules for that node. Some of the more common attributes, such as time_since_mastery_unit_x, would be available from a menu. Others may have to be collected by procedures written by the author.

The author also determines the types of the specified attributes. Following Michalski (1983) let us differentiate three types of attributes: nominal, ordered (which Michalski calls linear), and structured. Support for structured attributes, while probably an important feature for practical use, was not implemented because it is just a combination of the learning mechanisms for nominal and ordered attributes, and did not seem necessary for testing the fundamental behavior of ILP.

The typing of an attribute determines how ILP will treat the attribute when it generalizes and specializes descriptions. Determining the type of attributes is one way the author contributes background knowledge that biases the learning process. Determining attribute types is not a well defined task, and attributes may often seem to be on the borderline between types.

---

[21] An alternative to random selection is considered in the next chapter.

4.2.3 The description language

There may be several classification rules relating to each prerequisite (i.e., with the same right hand side). The disjunction of the left hand sides of these rules comprise the description for that prerequisite. It is this DNF description that authors can initialize, observe, and modify. A partial syntax of the description language is given concisely by the following Backus-Naur statements:

<description> ::= [<clause>] <cf> | [<clause>] <cf> v <description>
<clause> ::= <term> | <term> & <clause>
<term> ::= <attribute_name> = <attribute_value_range> | <attribute_name> = <attribute_value>
<attribute_value_range> ::= <attribute_value> .. <attribute_value>
<cf> ::= (<poscov> / <sum_of_poscov_negcov>)

An example of a description is:

[response_on_item_3 = sugar] (6/7) v [age = 4..5 & latency_on_item_3 = 3..8] (2/4)

Certainty factors are maintained by the system and are not entered or modified by the author. Attribute value ranges are permitted only on ordered attributes. Terms based on nominal attributes are called nominal terms, and those based on ordered attributes are called ordered terms.

4.2.4 The inductive learning procedure

From the author's viewpoint ILP maintains a current best description, called CURDESC, for each prerequisite. CURDESC is obtained by a beam search which is invoked immediately after a new example is acquired. The beam search starts with the previous CURDESC and applies generalization or specialization operations to discover new descriptions, one of which may be installed as the new CURDESC if it is better than the previous CURDESC. The author can initialize CURDESC and modify it at any time. If the author does not supply an initial description, CURDESC is initialized by the system to be a description with zero clauses.

The author has control over the search through four parameters which can be changed at any time:

- maximum beam width (max_width)
- maximum depth of search (max_depth)
- maximum number of clauses in description (max_clause)
- a weight used in the preference criterion which gives the cost of including one negative example relative to the cost of excluding one positive example (ω)

In general, the settings for max_width and max_depth will depend on the amount of available cpu time and fast memory. Theoretically, max_clause is the author's estimate of the maximum number of different clusters of students lacking the same prerequisite. But, as this is a difficult quantity to estimate, it may in practice simply reflect the author's taste for detail. A higher value of max_clause will tend to produce a more detailed description.The parameter ω allows the author to bias the trade-off between completeness and consistency in situations where both criterion cannot be satisfied.

In order to guide learning and calculate certainty factors, ILP maintains four lists of examples for each description:

PIN	containing pin_num positive examples included by the description
PEX	containing pex_num positive examples excluded by the description
NIN	containing nin_num negative examples included by the description
NEX	containing nex_num negative examples excluded by the description

Each member of PEX violates the completeness of the description, and each member of NIN violates its consistency. Descriptions are ranked during the search according to how well they minimize the cost:

$$\text{pex\_num} + (\omega \times \text{nin\_num})$$

Among descriptions with the same cost, those with fewer clauses are preferred in the interests of parsimony (p. 78). This simple measure for ranking descriptions is referred to as the **preference criterion**. Except in special cases where the author may place greater importance on consistency than completeness, ω should probably be set to one.

The structure of the description space searched by ILP is determined by two generalization operations and two specialization operations which ILP can invoke. Each operation takes as input a description and an example. Generalization operations are passed a positive example excluded by the given description, and specialization operations are passed a negative example included by the given description.

The generalization operations are:

```
GENERALIZE_CLAUSES (D : description, $e^+$ : example)
For each clause in D
    return one new description by making a copy of D and generalizing all terms in the
    clause which do not match $e^+$. Generalization is accomplished by deleting nominal
    terms and widening the value ranges of ordered terms just enough to include $e^+$.
    Ordered terms are dropped whenever their value ranges are extended to cover the
    entire value set;
```

```
CREATE_CLAUSE (D : description, $e^+$ : example)
Return one new description D' constructed as follows:
1) make a copy of D called D';
2) If the number of clauses in D' equals max_clause then delete the clause in D' with
minimal utility POSCOV - ($\omega \times$ NEGCOV);
3) Convert $e^+$ to a conjunctive clause and join it to D' by disjunction;
```

To illustrate how these operations work, suppose the current description D (without certainty factors) is:

$$[A=1 \ \& \ B=2] \vee [A=2 \ \& \ C=2..6]$$

where A and B are nominal attributes with the value set {1,2,3} and C is an ordered attribute. Suppose there is a positive example e^+ not covered by D:

$$(A=1, B=1, C=8)$$

Applying the GENERALIZE_CLAUSES operation will result in two new descriptions:

$$[A=1] \vee [A=2 \ \& \ C=2..6]$$

$$[A=1 \ \& \ B=2] \vee [C=2..8]$$

The GENERALIZE_CLAUSES operation combines the functions of Michalski's dropping conditions rule and closing interval rule. If support for structured attributes were provided, this operation would also encompass the climbing generalization tree rule.

Applying the CREATE_CLAUSE operation to D will result in one new description (assuming max_clause is greater than two):

$$[A=1 \ \& \ B=2] \vee [A=2 \ \& \ C=2..6] \vee [A=1 \ \& \ B=1 \ \& \ C=8]$$

The CREATE_CLAUSE operation serves the same function as Michalski's adding alternative rule, except that in the case where CREATE_CLAUSE must delete an old clause, it is possible, and in fact likely, that the new description will not be more general than D.

The specialization operations are:

```
SPECIALIZE_CLAUSES (D : description; $e^-$ : example)
For every clause in D covering $e^-$
    for every attribute $\alpha$
        If $\alpha$ is nominal and appears in term t of clause then
            for every value v of $\alpha$ not matching $e^-$ create one new description by making a
            copy of D and setting v as the value in term t;
        If $\alpha$ is nominal and does not appear in clause then
            for every value v of $\alpha$ not matching $e^-$ create one new description by making a
            copy of D and inserting in it a new term with attribute $\alpha$ and value v;
        If $\alpha$ is ordered and appears in term t of clause then create as many as two new
            descriptions by making copies of D and raising the lower bound and/or
            lowering the upper bound of the value range of t just enough to exclude $e^-$ ;
        If $\alpha$ is ordered and does not appear in clause then create as many as two new
            descriptions by making copies of D and inserting in each a new term with
            attribute $\alpha$ and a value range just excluding $e^-$ ;
```

```
DELETE_CLAUSE (D : description; $e^-$ : example)
For every clause in D covering $e^-$ return one new description which excludes E by deleting
    the clause;
```

Again, suppose the current description D is:

[A=1 & B=2] ∨ [A=2 & C=2..6]

And suppose there is a negative example e^- covered by D:

(A=2, B=2, C=4)

Applying the SPECIALIZE_CLAUSES operation will result in several new descriptions in which the second clause is specialized to exclude e^- :

[A=1 & B=2] ∨ [A=2 & C=5..6]
[A=1 & B=2] ∨ [A=2 & C=2..3]
[A=1 & B=2] ∨ [A=1 & C=2..6]
[A=1 & B=2] ∨ [A=3 & C=2..6]
[A=1 & B=2] ∨ [A=2 & B=1 & C=2..6]
[A=1 & B=2] ∨ [A=2 & B=3 & C=2..6]

Note that, in cases where the negative example is covered by more than one clause, the new description as a whole will still cover e^- . In such cases, further specialization operations must be applied to exclude e^- . Even with this restricted kind of specialization, SPECIALIZE_CLAUSES tends to generate considerably more new descriptions than GENERALIZE_CLAUSES. In order to avoid the rapid proliferation of descriptions by this operation, the author is recommended to minimize the number of attributes and the size of the value sets of nominal attributes.

Applying the DELETE_CLAUSE operation to D will result in one new description:

[A=1 & B=2]

All descriptions created in the search have exactly the same set of examples as CURDESC, although they may be partitioned between the four example lists differently. For each description, examples in the PEX and NIN lists have a flag indicating whether they have been tried, that is, have served as the basis for generalization or specialization operations. When an example is used for generalization or specialization it is set to TRIED. When a new description is created all its examples are initialized to NOT_TRIED. An example will only be used for a generalization or specialization operation if it is NOT_TRIED. The TRIED/NOT_TRIED flag enables ILP to avoid searching a region of the description space which has already been searched, unless new conditions warrant it.

For the sake of simplicity, it is assumed that ILP and the classification rules it learns reside in a central computer connected to student workstations or terminals. From time to time the workstations request recommendations from the classification rules. The workstations do the recommended tests and transmit the results back to ILP in the form of classified examples. One student may generate several examples, one for every prerequisite on which he is tested. Each example is pushed on to the front of one of the example lists (PIN, PEX, NIN, NEX) maintained by the CURDESC of the prerequisite on which the example is classified (recall that there is one CURDESC for every prerequisite). When a new example is pushed on to an example list, it is set to NOT_TRIED.

ILP is activated after a set of examples generated by a student is received and assigned to the example lists. If subsequent examples are received while the learning process triggered by the first set is still active, they wait until ILP is finished before being assigned to example lists and activating ILP again.

ILP examines every CURDESC, including those which did not receive new examples, to judge whether cpu time should be invested in trying to improve them. The learning process is applied to all CURDESCs with NOT_TRIED examples in their PEX or NIN lists. In trying to improve each of these CURDESCs, ILP makes use of two lists of descriptions: DESCLIST which is initialized with the CURDESC as its first and only member; and TEMPDESCLIST, which serves as temporary storage for new descriptions. ILP makes a number of passes through DESCLIST, in which the following procedure IMPROVE_DESCRIPTION is applied to each of its members:

```
IMPROVE_DESCRIPTION (D : description)
If D has at least one NOT_TRIED example in its PEX list and none in its NIN list then do
GENERALIZE(D)
Else if D has at least one NOT_TRIED example in its NIN list but none in its PEX list then do
SPECIALIZE(D)
Else if D has NOT_TRIED examples in both its PEX list and its NIN list then
    If pex_num $\geq \omega \times$ nin_num do GENERALIZE(D) else do SPECIALIZE(D);
```

```
GENERALIZE (D : description)
Let $e^+$ be the first NOT_TRIED example in the PEX list of D;
Set $e^+$ to TRIED;
Do CREATE_CLAUSE(D, $e^+$) and put the resulting description in TEMPDESCLIST;
Do GENERALIZE_CLAUSES(D, $e^+$) and put the resulting descriptions in TEMPDESCLIST;
```

```
SPECIALIZE (D : description)
Let $e^-$ be the first NOT_TRIED example in the NIN list of D;
Set $e^-$ to TRIED;
Do DELETE_CLAUSE(D,$e^-$) and put the resulting description in TEMPDESCLIST;
Do SPECIALIZE_CLAUSES(D,$e^-$) and put the resulting descriptions in TEMPDESCLIST;
```

Each pass through DESCLIST is one level of the beam search. After each pass, TEMPDESCLIST is merged into DESCLIST. Then, after being sorted according to the preference criterion, the new DESCLIST is truncated to contain no more than max_width members (see Appendix A). The beam search halts when max_depth passes have been completed, or prior to that if there are no NOT_TRIED examples in the PEX or NIN lists of any of the descriptions in DESCLIST. The first description in DESCLIST becomes the new CURDESC.

To summarize, ILP starts a beam search with a single description which is either the result of a previous learning episode, or has been entered directly by the author. The search is determined by pruning according to the preference criterion, and also by a homeostatic heuristic which maintains a balance (calibrated by the ω parameter) between positive examples excluded and negative examples included by the description. For instance, when the positive examples excluded outweigh the negative examples included by a description, this heuristic forces the search into more general regions of the description space.

ILP is designed to learn disjunctive descriptions, tolerate noisy examples, and deal with multi-valued and ordered attributes. The most prominent feature distinguishing ILP from previous learning programs is its degree of emphasis on incremental refinement of a user-supplied description. The need to support incremental refinement has led to the abandonment of the positive example deletion strategy used extensively by previous learning programs, and to the adoption of a strategy which biases learning with hypotheses entered by the author. It is possible to imagine cases where the author's prior knowledge is so poor that the advantages of this approach are nullified. Ultimately, the best learning procedure for the instructional application described here will depend on the prevalence of such cases.

4.3 References

Articles referenced as *Machine learning I* or *Machine learning II* appeared in one of the following:

Michalski, R., Carbonell, J., & Mitchell, T. (Eds.). (1983). *Machine learning: An artificial intelligence approach* (Vol. 1). Los Altos CA: Kaufmann.

Michalski, R., Carbonell, J., & Mitchell, T. (Eds.). (1986). *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos CA: Kaufmann.

Bruner, J., Goodnow, J., & Austin, G. (1956). *A study of thinking*. New York: Wiley.

Bundy, A., Silver, B., & Plummer, D. (1985). An analytical comparison of some rule-learning programs. *Artificial Intelligence*. 27, 137-181.

Burton, R. (1982). Diagnosing bugs in a simple procedural skill. In D. Sleeman, J. S. Brown (Eds.), *Intelligent tutoring systems*. London: Academic Press.

Carbonell, J., Michalski, R., & Mitchell, T. (1983). An overview of machine learning. *Machine learning I*.

Dietterich, T. (1982). Learning and inductive inference. In R. Cohen & E. Feigenbaum (Eds.), *The Handbook of Artificial Intelligence* (Vol. 3). Los Altos CA : Kaufmann.

Dietterich, T., & Michalski, R. (1983). A comparative review of selected methods for learning from examples. *Machine learning I*.

Feingold, S. (1968). Planit - a language for CAI. *Datamation*. 14 (9), 41-48.

Hayes-Roth, F., & McDermott, J. (1976). Learning structured patterns from examples. *Proceedings of the third international joint conference on pattern recognition*. p. 419-423.

Hunt, E., Marin, J., & Stone, P. (1966). *Experiments in induction*. New York: Academic Press.

Kalish, D., Montague, R., & Mar, G. (1980). *Logic: Techniques of formal reasoning*. New York: Harcourt Brace Jovanovich.

Kimball, R. (1982). A self-improving tutor for symbolic integration. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems*. London: Academic Press.

Langley, P. (1987). A general theory of discrimination learning. In D. Klahr, P. Langley, & R. Neches (Eds.), *Production system models of learning and development*. Cambridge MA: MIT Press.

Langley, P., Wogulis, J., & Ohlsson, S. (1987). Rules and principles in cognitive diagnosis. In N. Fredericksen (Ed.). *Diagnostic monitoring of skill and knowledge acquisition*. Hillsdale NJ: Erlbaum.

Lee, W., & Ray, S. (1986a). *Probabilistic rule generator: A new methodology of variable-valued logic synthesis*. Report No. UIUCDCS-R-86-1264. Department of Computer Science, University of Illinois at Urbana-Champaign.

Lee, W., & Ray, S. (1986b). Rule refinement using the probabilistic rule generator. *Proceedings of AAAI-86*, p. 442-447.

Lenat, D. (1983). The role of heuristics in learning by discovery: Three case studies. *Machine learning I*.

McCulloch, W., & Pitts, W. (1943). A logical calculus of ideas imminent in nervous activity. *Bulletin of Mathematical Biophysics*, 5, 115-133.

Michalski, R. (1973). Discovering classification rules using variable-valued logic system VL_1. *Proceedings of the Third International Joint Conference on Artificial Intelligence*. p. 162-172.

Michalski, R., & Chilausky, R. (1980). Learning by being told and learning from examples: An experimental comparison of the two methods of knowledge acquisition in the context of developing an expert system for soybean disease diagnosis. *International Journal of Policy Analysis and Information Systems*, 4(2), 125-161.

Michalski, R. (1983). A theory and methodology of inductive learning. *Artificial Intelligence*, 20, 111-161.

Michalski, R. (1986). Understanding the nature of learning: Issues and research directions. *Machine learning II*.

Minsky, M., & Papert, S. (1969). *Perceptrons*. Cambridge MA: MIT Press.

Mitchell, T. (1977). Version spaces: A candidate elimination approach to rule learning. *Proceedings of the Fifth International Joint Conference on Artificial Intelligence*. p. 305-310.

Mitchell, T. (1982). Generalization as search. *Artificial Intelligence*. 18, 203-226.

Mitchell, T., Utgoff, P., & Banerji, R. (1983). Learning by experimentation: Acquiring and refining problem-solving heuristics. *Machine learning I*.

Nilsson, N. (1965). *Learning machines*. New York: McGraw-Hill.

O'Shea, T. (1982). A self-improving quadratic tutor. In D. Sleeman & J. S. Brown (Eds.), *Intelligent tutoring systems*. London: Academic Press.

Popper, K. (1959) *The logic of scientific discovery*. New York: Basic Books.

Quinlan, J. R. (1983). Learning efficient classification procedures and their application to chess end games. *Machine learning I*.

Quinlan, J. R. (1986). The effect of noise on concept learning. *Machine learning II*.

Rosenblatt, F. (1958). The perceptron: a probabilistic model for information storage and organization in the brain. *Psychological Review*, 65, 386-407.

Rumelhart, D., & McClelland, J. (1986). *Parallel distributed processing: Explorations in the microstructure of cognition*. Cambridge, MA: MIT Press.

Simon, H., & Lea, G. (1974). Problem solving and rule induction: a unified view. In L. Gregg (Ed.), *Knowledge and cognition*. Hillsdale NJ : Lawrence Erlbaum.

Spiegelhalter, D. (1986). A statistical view of uncertainty in expert systems. In W. Gale (Ed.) *Artificial intelligence and statistics*. Reading, MA: Addison-Wesley.

Kedar-Cabelli, S., & Mahadevan, S. (1986). Bibliography of recent machine learning research. *Machine learning II*.

Utgoff, P. (1986). Shift of bias for inductive concept learning. *Machine learning II*. 107-148.

Utgoff, P., & Nudel, B. (1983). Comprehensive bibliography of machine learning. *Machine learning I*.

VanLehn, K. (1987). Learning one subprocedure per lesson. *Artificial Intelligence*, 31(1), 1-40.

Winston, P. (1975). Learning structural descriptions from examples. In P. Winston (Ed.) *The psychology of computer vision*. New York: McGraw-Hill.

Chapter 5

Performance of the Inductive Learning Procedure

The purpose of this chapter is to present information on the performance of ILP under various controlled conditions. This information provides a better understanding of the workings of ILP, and suggests some guiding principles for its use in real instructional environments. The chapter also proposes several improvements to ILP aimed at correcting deficiencies uncovered by this evaluation.

ILP was implemented in Pascal on a VAX 11/780 minicomputer running the VMS operating system, with maximum virtual memory size set to about 11 megabytes. The VAX 11/780 is usually considered to have a speed of about 1 MIPS. Computers of this class are now common in even the smaller post-secondary educational institutions (Hunka, 1987). Current trends indicate that, within a few years, computers of somewhat greater power will be widely available.

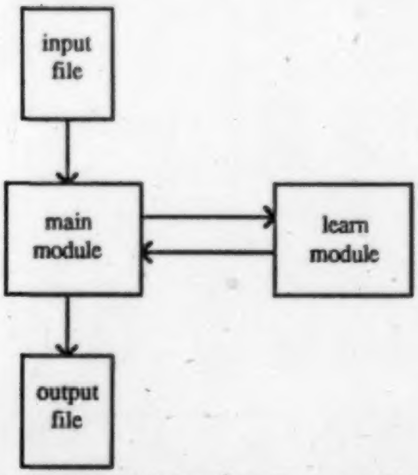


Figure 5.1 Modules and files developed for performance evaluation.

Figure 5.1 shows the program modules and data files that were active at run time. The main module initialized the program with input data, managed file I/O, and simulated the recommendation and testing phases described in the previous chapter. The learn module dealt with the learning phase, which is the main focus of this investigation. The input file contained the user-supplied parameters:

- max_width (maximum beam width)
- max_depth (maximum depth of search)
- max_clause (maximum number of clauses in description)
- ω (the cost of including one negative example relative to the cost of excluding one positive example)

as well as definitions of the attributes, initial descriptions, and a set of categorized examples. The output file saved a trace of the performance of the program, primarily serving as a record of incorrect recommendations.

5.1 A close-up view of ILP at work

A visual interpretation of the behavior of ILP can be presented for problems involving two ordered attributes. In this interpretation, a concept description is depicted as a set of rectangles occupying a two dimensional example space. Each rectangle corresponds to a clause in the description language representation.

This close-up view is useful for confirming that ILP behaves in a reasonable manner on some very simple learning tasks. For instance, where positive and negative examples are linearly separable, we expect to see ILP form a description approximately covering a region bounded by a linear discriminant function. This problem was posed to ILP by assuming two concepts P1 and P2, and two ordered attributes X and Y both with the value set $\{0,1,\ldots,99\}$. 100 examples of the form (x,y) were uniformly scattered throughout the example space. Examples satisfying $x > y$ were assigned to P1 and all others were assigned to P2.

Although in this case the concepts P1 and P2 are (somewhat unnaturally) mutually exclusive, recall that ILP does not assume that they will be. This means that unless an incorrect recommendation is made, ILP will not recognize that a positive example for one prerequisite is a negative example for the other.

The categorized examples were sequentially fed to ILP in random order. The parameters were set as follows: max_width = 12, max_depth = 6, max_clause = 10, ω = 1.

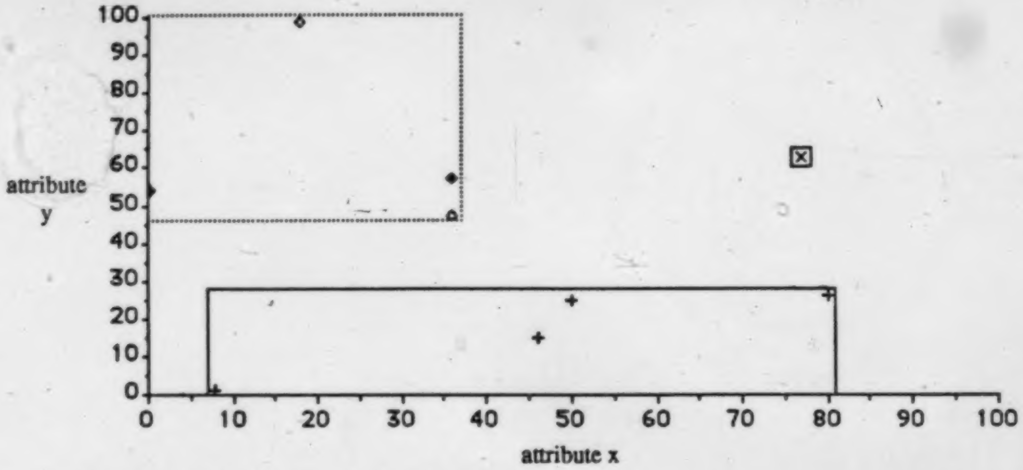


Figure 5.2 Descriptions learned after 9 examples.

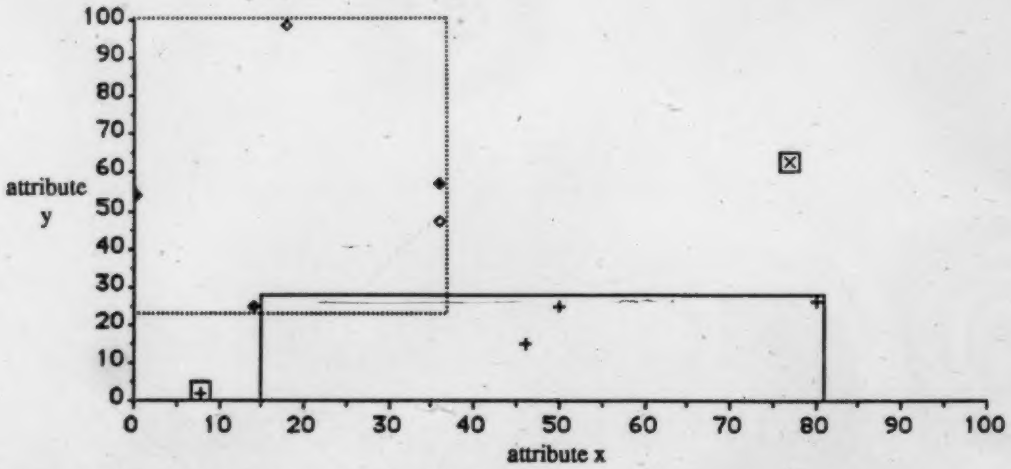


Figure 5.3 Descriptions learned after 10 examples.

Figures 5.2 through 5.5 depict the descriptions developed by ILP after 5, 10, 56, and 100 examples had been encountered. Clauses in the P1 description are represented as dotted rectangles and clauses in the P2 description appear as solid rectangles. Positive examples of P1 are shown as diamonds. The filled diamonds indicate positive examples of P1 which were also discovered to be negative examples of P2. Positive examples of P2 are shown as crosses (+ and ×). The × are positive examples of P2 which were also discovered to be negative examples of P1.

Figure 5.2 shows the two current descriptions (CURDESCs) held by ILP after processing the 9th example. P1 has one clause and P2 has two clauses. Notice that the smaller clause of P2 covers only the single point (77,63).

The 10th example (14,25) falls within the large clause of P2 but, after an incorrect recommendation, turns out to belong to P1. As shown in Figure 5.3, this example causes a generalization of the single P1 clause. The large P2 clause is specialized to exclude the new example, but in so doing must exclude an old example at (8,2). This situation is resolved at the next level of the search by creating a new clause to cover (8,2).

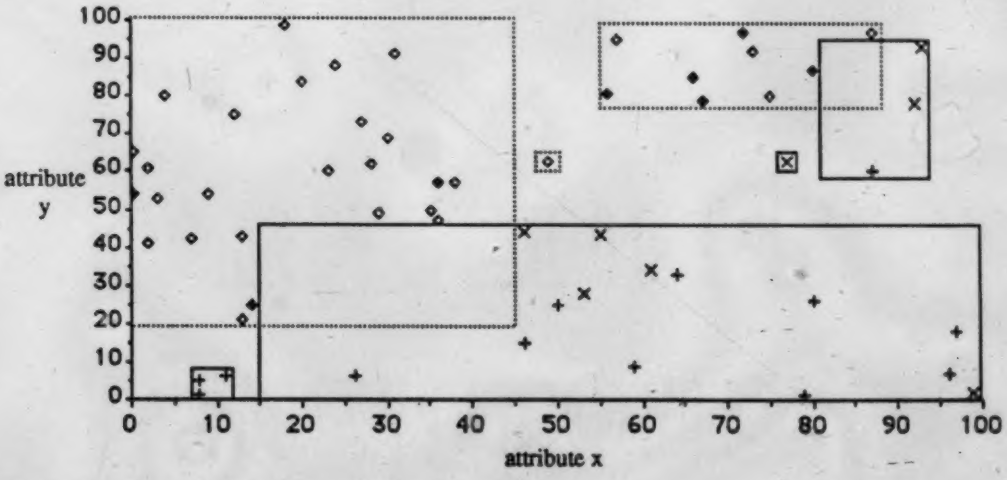


Figure 5.4 Descriptions learned after 56 examples.

Figure 5.4 shows the CURDESCs after the 56th example has been processed. By this time, the P2 clause in the lower left that was created after the 10th example has been generalized to accommodate recent examples. The 56th example at (49,61) forced ILP to create a new clause for P1 because neither of the two existing clauses could be generalized to cover it without also covering negative examples.

Figure 5.5 shows what ILP learned from the entire example set. Both final descriptions satisfy the criteria of completeness and consistency, as did all CURDESCs developed throughout the run. The description for P2 achieves this with the fewest possible clauses. The description found for P1 contains four clauses:

[x=0..41 & y=33..99] v

[x=49..87 & y=76..99] v

[x=2..14 & y=15..25] v

[x=49..66 & y=63..68]

but there exists a solution requiring only three clauses:

[x=0..25 & y=15..99] v

[x=25..76 & y=46..99] v

[x=76..87 & y=87..99]

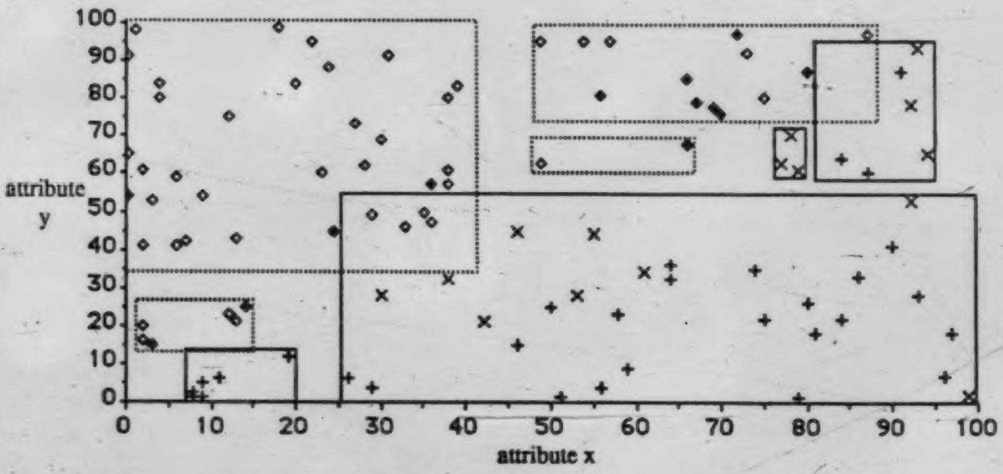


Figure 5.5 Descriptions learned after 100 examples.

The empty diamonds occupying the intersection of the two large clauses were present before a generalization of P2 covered them. Thus, they were never the subject of incorrect recommendation, and are not known to P2 as negative examples.

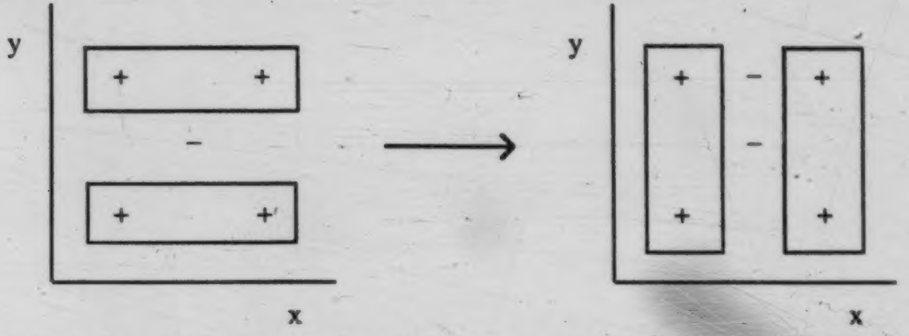


Figure 5.6 A case where a single example produces a radical change in the description.

Especially when the number of known examples is small, a single example may produce a radical change in the description. Figure 5.6 shows such an event observed with a description having two clauses. Positive and negative examples are indicated by + and -. This illustrates the point that although ILP is biased by the existing description, it will jump to the best description discovered in the search, even if it is a significant departure from the existing description. In this case, the solution discovered by ILP was found after a search four levels deep. In the first level, the top clause was specialized to exclude the new negative example, resulting in the exclusion of the positive example in the top left. In the second level, the bottom clause was generalized to cover the recently excluded positive example, but in so doing was forced to cover the entire description space. In the third level, this large clause was specialized, thus forming the final clause on the left side. In the fourth level, the small clause covering the positive example in the top right was generalized to form the final clause on the right side.

5.2 Four simulated cases for testing ILP

Four sets of examples (i.e., categorized student models) were created to test the performance of ILP. Each case is fully documented in Appendix B. The cases vary in realism from Case 1, which is little more realistic than the linearly separable examples seen in the last section, to Case 4, which is perhaps the most realistic and difficult of the four learning tasks. Although these artificial cases are necessary for preliminary and formative evaluation of ILP, they are not regarded as a substitute for field tests with real students and courseware. The cases represent attempts to simulate instructional situations for the purpose of: (1) catching any unanticipated behavior of ILP, (2) providing an empirical basis for improvements to ILP, (3) determining practical parameter settings, and (4) suggesting ways that the author can most successfully interact with ILP.

Unless otherwise stated, the tests reported here are initialized with a CURDESC containing zero clauses. The primary measure of performance is the number of incorrect recommendations made over the entire set of examples. Incorrect recommendations include two kinds of events: (1) a prerequisite test is recommended which is subsequently passed, and (2) a student is returned to the objective with which he was having difficulty while still lacking one or more prerequisites. Errors of the first kind were always more prevalent because the second kind of error could only occur when a student lacked more than one prerequisite.

Case 1 assumed six binary attributes $a_1, a_2, ..., a_6$. These were nominal attributes having the value set $\{0,1\}$. Thus the entire example space allowed for only $2^6 = 64$ distinct examples. There were three prerequisites P1, P2, and P3, each with a rather arbitrarily constructed target description. For instance, the target description for P2 was:

$$[a_2=1 \ \& \ a_3=0 \ \& \ a_4=0] \vee [a_2=1 \ \& \ a_5=1 \ \& \ a_6=1]$$

The other target descriptions were of similar complexity. Altogether these target descriptions covered 29 distinct examples, about half the example space.

Fifty examples were obtained by (1) using a random number generator to produce attribute vectors uniformly scattered over the example space, and (2) selecting only those vectors which matched one or more of the target descriptions. In effect, the target descriptions served as a filter, rejecting all student models which ILP would not encounter.

The selected attribute vectors were categorized according to which target descriptions they matched.

Unlike subsequent cases, the first 29 examples of Case 1 were sampled without replacement. This departure from realism was allowed in order to more clearly demonstrate that ILP's performance exceeds that expected from simple rote learning. A rote learning procedure stores all examples encountered but does no generalization. Therefore, it exhibits improved performance only on repeated examples.

Figure 5.6 shows the cumulative number of incorrect recommendations (errors) after each example encountered. The solid plot records the performance of ILP over the course of one run. The dotted plot records the performance of a non-learning procedure initialized with the same zero-clause descriptions as ILP. Because the non-learning procedure could not modify these initial descriptions, its recommendations were based entirely on chance. There was no substantial variation between different runs of these procedures. As with all presentations in this chapter, the runs shown here are typical.

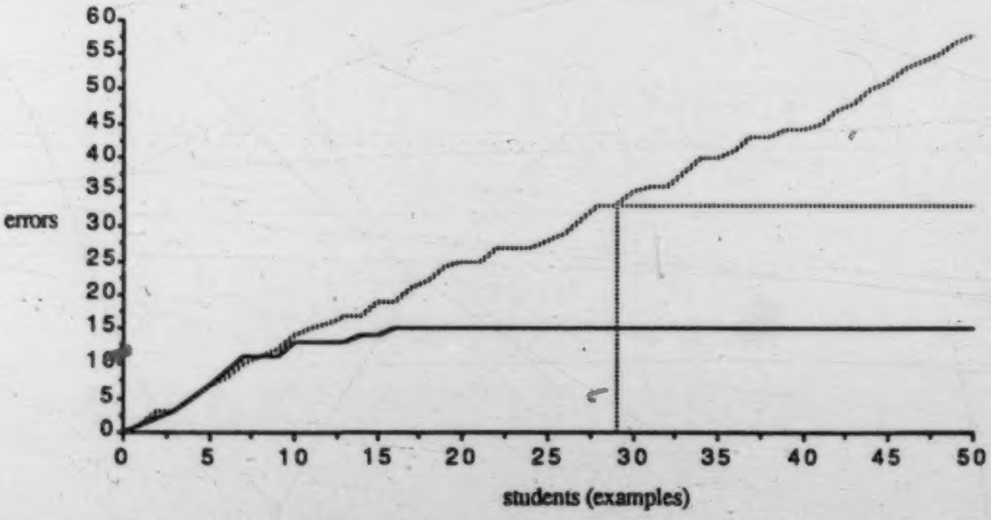


Figure 5.7 Performance of ILP on Case 1.

The vertical dotted line (x=29) indicates the point at which all distinct examples have been encountered. After this point a rote learning procedure would perform perfectly, as indicated by the horizontal dotted line. However, before this point it would perform no better than the non-learning procedure.

ILP learned a correct set of descriptions by the 17th example. The output file also showed that all CURDESCS produced throughout the learning process were consistent and complete with regard to examples already encountered.

Were the descriptions discovered by ILP the same as the target descriptions? The description learned for P2 was identical to the target description. The other two were different from, but logically equivalent to, their target descriptions. For instance, the target description for P1 was:

$$[a_1=1 \ \& \ a_2=1 \ \& \ a_3=1] \vee [a_1=1 \ \& \ a_2=1 \ \& \ a_4=1]$$

The description learned was:

$$[a_1=1 \ \& \ a_2=1 \ \& \ a_3=1] \vee [a_1=1 \ \& \ a_2=1 \ \& \ a_3=0 \ \& \ a_4=1]$$

The equivalence of these two expressions can be shown by propositional logic.

In general though, we cannot expect a correct solution to be equivalent to the target description because the negative examples encountered by ILP are restricted. In order to be a negative example for one target description, an attribute vector must be a positive example for one of the other target descriptions. For these reasons and others, comparisons between target descriptions and learned descriptions do not provide a useful or valid measure of the performance of ILP.

In an attempt to achieve greater realism in Case 2, a specific content was assumed for the primary objective. Multiple column addition with two addends was chosen as the objective because student error patterns in learning this task are understood relatively well. For precisely this reason however, ILP would not be as useful in this domain as in domains which have not been studied.

A rudimentary hierarchical analysis was performed on the addition task. It was broken down into three sequential steps which start with the right-most column and are repeated on each column until the addition is complete. Objectives teaching these steps became the prerequisites of the primary objective. Tests of mastery were posited for the primary objective and each of the prerequisites. Any student not correctly answering all six questions of the primary objective test was passed to the recommendation phase.

In many domains, and particularly in arithmetic, raw student responses are the best attributes on which to base predictions of prerequisite failure. So in this case, responses to the six items in the primary objective test were set up as the attributes to be used by ILP (see Appendix B). They were specified as ordered attributes, mainly because the size of their value sets {0, 1,...9999} makes them impractical for use as nominal attributes.

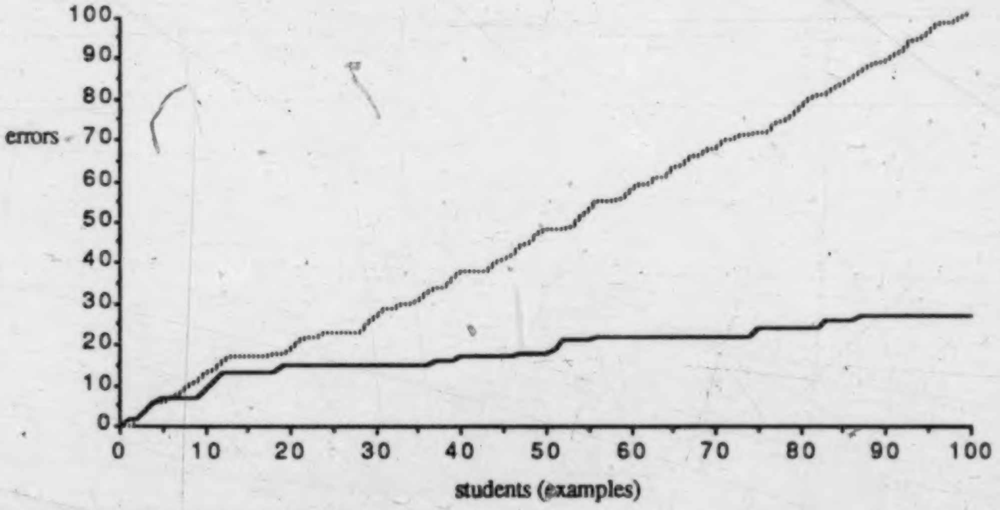


Figure 5.8 Performance of ILP on Case 2.

100 attribute vectors were generated with the simplifying assumption of only one error pattern, or bug, per student. The bugs were four common error patterns (Ashlock, 1976, p. 20) and an "addition facts" bug in which students forget a few single digit sums. Three of the common error patterns were interpreted as failures of prerequisite P1, one error pattern was interpreted as a failure of P3, and the addition facts bug as a failure of P2. Although this case is in some respects more realistic than the other cases, without the presence of noise it is a fairly easy learning task. This is so because all error patterns except

the addition facts bug produce only one distinct attribute vector. A random process generated 46 examples of the addition facts bug, of which 43 were distinct. The remaining 54 examples were divided roughly equally between the four error patterns.

Figure 5.8 shows the performance of ILP on Case 2. As before, the solid line plots the incorrect recommendations (errors) accumulated by ILP and the dotted line plots incorrect recommendations of a non-learning procedure. ILP had no difficulty in generating complete and consistent CURDESCs after each example encountered. However, because the attributes were specified as ordered, there was an inappropriate generalization over the three error patterns of P1. This resulted in (1) incorrect recommendation of new negative examples covered by the general clause (2) a rather misleading description for P1. Here is a case where the trivial disjunction (i.e., the disjunction of the three error patterns) is the most appropriate description. The problem is that the number of possible responses to an item is far too large for ILP to deal with the item as a nominal attribute unless the author can find some way of grouping the responses into a few categories.

Some of the ordered attributes hypothesized for Cases 3 and 4 would be expected to have non-uniform distributions. To simulate these kinds of attributes, procedures were written to generate random numbers falling in either normal or skewed (chi-square) distributions. Normal attributes were specified by mean and standard deviation. Skewed attributes were specified by mode and degrees of freedom. For instance, in Case 3, the attribute TSMP1 (time since mastery of prerequisite 1) was specified as a skewed attribute with a mode of 5 hours and 2 degrees of freedom. Attributes such as verbal aptitude and age were hypothesized to have normal distributions.

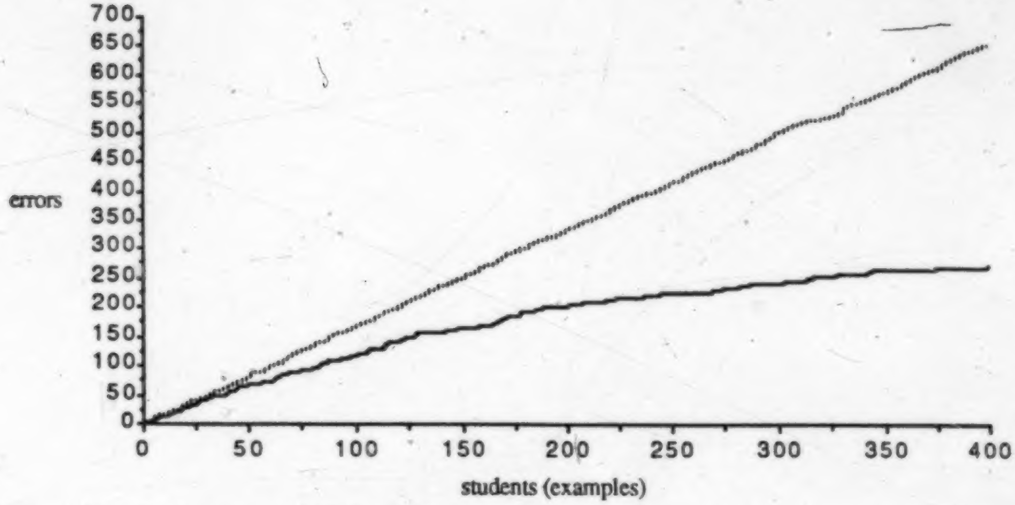


Figure 5.9 Performance of ILP on Case 3.

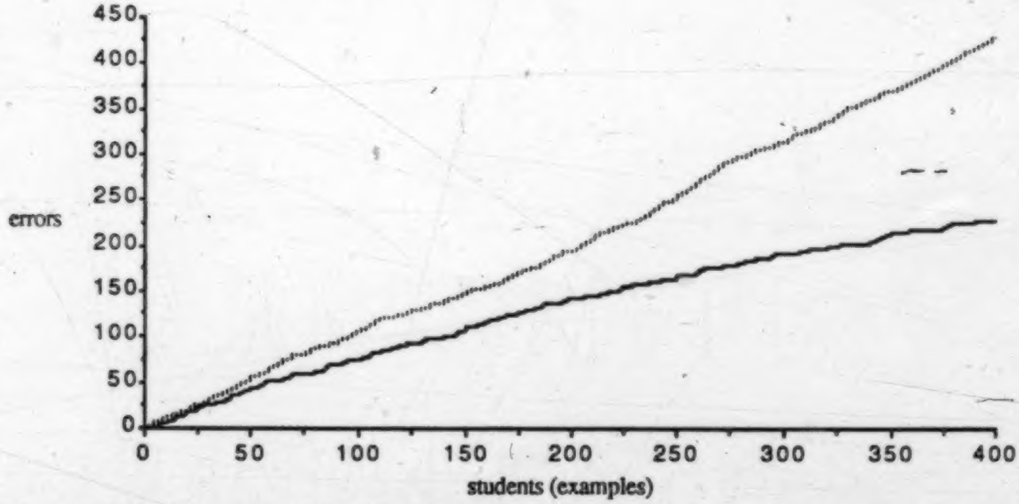


Figure 5.10 Performance of ILP on Case 4.

The input files for Cases 3 and 4 both contained 400 examples. Case 3 assumed four prerequisites and 12 ordered attributes. Case 4 assumed three prerequisites and 15 attributes (6 nominal and 9 ordered). The performance of ILP on these cases is shown in Figures 5.9 and 5.10.

5.3 Pre-generalization

In the version of ILP described in Chapter 4, a new example not matched by any CURDESC was subject to a random recommendation. But it often happens that such an example is "closer" to one CURDESC, and turns out to be a positive example resulting in a generalization of that CURDESC. An improvement was sought which would recommend that the non-matching student be tested on the prerequisite most likely to result in failure. Euclidean distance (in the example space) between the CURDESC and the example is not a valid criterion because it fails to account for intervening negative examples.

After some experimentation, a heuristic was developed which recommends the prerequisite whose CURDESC is most easily generalized to cover the non-matching example. When a clause is generalized with the GENERALIZE_CLAUSES operation, it covers one or more positive examples that were not previously covered by any clauses in the same CURDESC. Such examples are transferred from the PEX list to the PIN list. Let us denote the number of these examples as POS_GAINED. This generalization may also force the clause to cover negative examples which the clause did not previously cover. Let us denote the number of such negative examples as NEG_GAINED. In pre-generalization, the GENERALIZE_CLAUSES operation is temporarily applied to all CURDESCs to cover the non-matching example. The prerequisite is recommended whose CURDESC has the clause scoring highest on the following criterion:

$$\frac{\text{POS\_GAINED}}{\text{POS\_GAINED} + \text{NEG\_GAINED}}$$

A secondary advantage of pre-generalization is that it frees the behavior of ILP from random fluctuation. Runs with the identical input produce an identical result.

A simple experiment was performed to verify that pre-generalization does produce fewer incorrect recommendations. For each case, ILP was run six times with no pre-generalization (npg) and once with pre-generalization (pg). Parameter settings were the same as for the runs presented in the previous section. Table 5.1 shows the mean and standard deviation of the number of incorrect recommendations obtained in npg runs, the number obtained in the pg run, and the difference between $\overline{\text{npg}}$ and pg expressed in standard deviation units. Pre-generalization had little effect on Case 1, but produced substantial improvement with the other cases. This result was considered good enough to incorporate pre-generalization in all subsequent tests.

Table 5.1 A comparison of number of incorrect recommendations with pre-generalization (pg) and without pre-generalization (npg).

	$\overline{\text{npg}}$	sd_{npg}	pg	$(\overline{\text{npg}} - \text{pg})/\text{sd}_{\text{npg}}$
Case 1	14.3	1.0	15	-0.7
Case 2	23.8	3.3	15	2.7
Case 3	258.8	13.1	205	4.1
Case 4	218.2	18.2	170	2.7

5.4 The effect of varying search parameters

An early version of ILP did much less searching of the description space. It functioned in about the same way as the current version does when max_depth = 1. To verify that the greater search allowed by the current version produces fewer incorrect recommendations, each case was tested with varying max_width and max_depth parameters. Some of the results are shown in Table 5.2.

Table 5.2 Effect of increasing the search on number of incorrect recommendations.

	max_depth=1	max_depth=12 max_width=12
Case 1	19	15
Case 2	17	15
Case 3	199	205
Case 4	233	170

Capability for more than one level of search was a substantial benefit only in Case 4. Figure 5.11 is presented as evidence that this benefit tends to increase continuously as the search capability increases. A very similar pattern was observed in Case 1. In all cases when max_width=12, complete and consistent CURDESCs were always found within a depth of 6 levels, and therefore ILP never pursued the search deeper than 6 levels. Although it is unusual that more extensive search led to slightly worse performance with Case 3, it should be expected to occur occasionally. The reason is that the performance of ILP on any case is determined by a sequence of many searches, each one terminating with the selection of a single optimum description (CURDESC). In processing one of the early examples, it may happen that a wider search uncovers a current optimum, thus causing ILP to abandon a region of the description space favored by later examples.

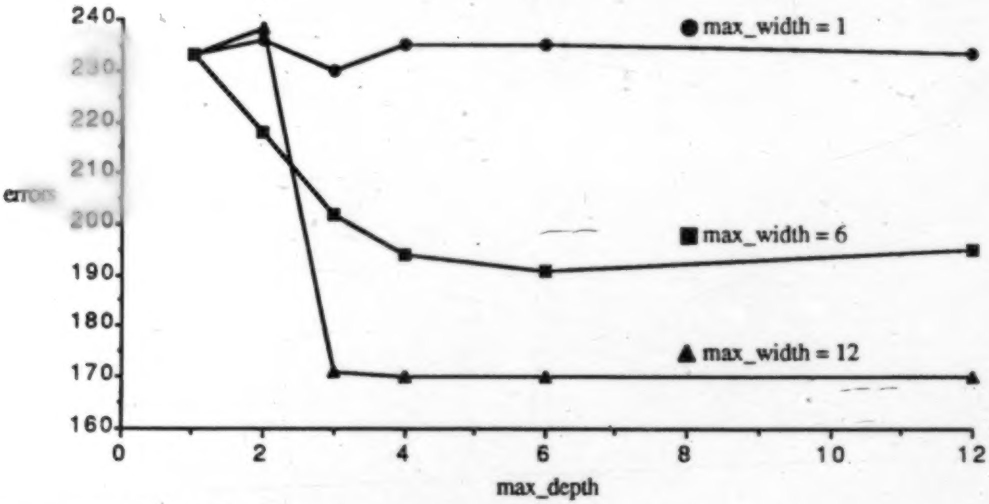


Figure 5.11 Varying the search parameters with Case 4.

5.5 The effect of noise

The introduction of noise into the examples always has the effect of degrading the performance of a learning program. In programs which assume all examples to be valid, the effect of noise can be catastrophic because a single invalid example may contradict and defeat a description which has been built upon many valid examples.

In order to investigate how ILP fares in the presence of noise, a program was written which corrupted input files with a given n percent of noise. This means that the program altered, with a probability of n/100, attribute values comprising examples in the input file. For instance, if n=10 and there are 10 attribute value-pairs in each example, then we expect one corrupted attribute-value pair per example. Substituted values were randomly chosen from the value set according to the distribution from which the original values were generated. Unlike Quinlan's (1986) procedure, the substituted value was required to be different from the original value. Noise was not introduced into the example classifications because this would have required introducing several complications and further assumptions into the simulation of the recommendation phase.

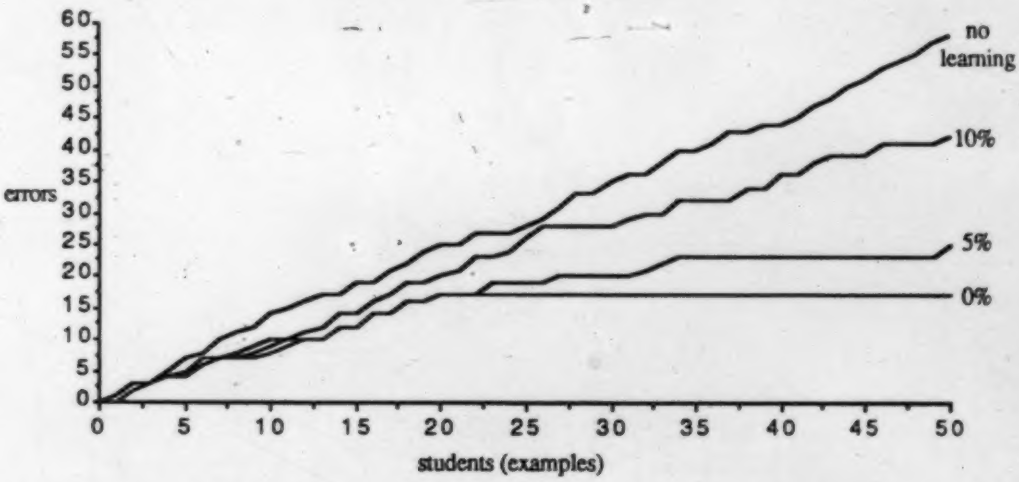


Figure 5.12 Performance of ILP on Case 1 with 0%, 5%, and 10% noise.

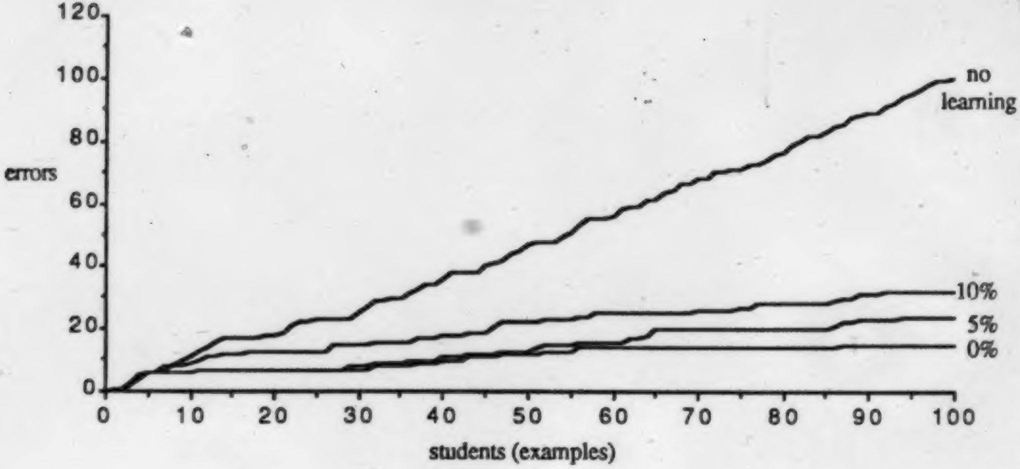


Figure 5.13 Performance of ILP on Case 2 with 0%, 5%, 10% noise.

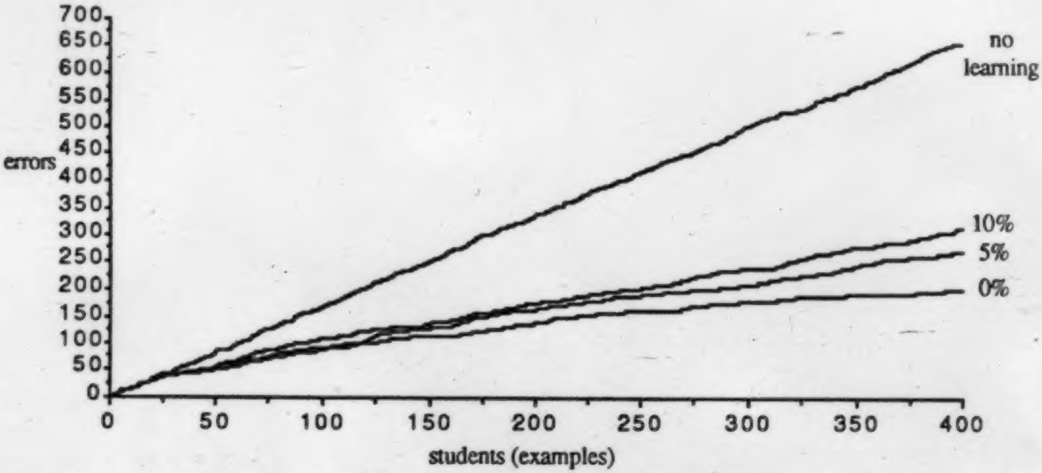


Figure 5.14 Performance of ILP on Case 3 with 0%, 5%, and 10% noise.

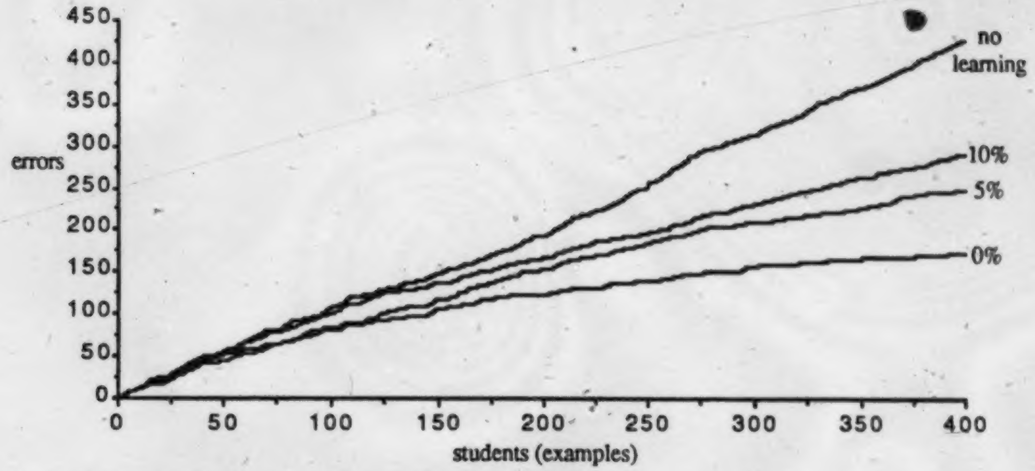


Figure 5.15 Performance of ILP on Case 4 with 0%, 5%, and 10% noise.

Initial tests on noisy versions of Case 4, with max_depth=6, max_width=12, and max_clause=10, occasionally exceeded memory capacity as the number of examples became large (>300). Thus in all subsequent sessions, the search parameters were set at max_depth=4, max_width=8, and max_clause=6. Figures 5.12 through 5.15 show how the performance of ILP deteriorates with increasing levels of noise.

5.6 Initialization of descriptions

Further tests found that, with noisy examples, description initialization provided only limited benefit. As expected, when the CURDESCs were initialized to be the same as the target descriptions, there was a substantial reduction in the number of incorrect recommendations. However, much smaller benefits were obtained when the CURDESCs were initialized with only partial reproductions of the target descriptions. For instance, with Case 4 (5% noise) ILP produced 160 incorrect recommendations when CURDESCs were initialized with target descriptions, as compared with 250 under the default condition. When initialized with single clauses from the target descriptions, 218 incorrect recommendations were produced. Initialization with single terms from the target descriptions, perhaps the most realistic scenario, produced the same result as the default condition.

The reason for the limited effect of initial descriptions is that they are easily deformed at the beginning of the run by a small number of invalid examples. Descriptions entered later in the run derive stability from valid examples already encountered, to the extent that they agree with those examples.

5.7 Other observations

The current implementation showed that a 1 MIPS machine with about 11 megabytes of memory will allow, at every level of the beam search, about 1000 descriptions to be created and simultaneously maintained in memory for sorting (assuming 400 examples). Since every description maintains its own complete set of examples this limit will decrease as the program acquires more examples[22]. In order for the preference criterion to properly guide the search, it is important that the descriptions spawned at each level not exceed memory capacity. In all cases tested here the search parameters max_clause=6, max_width=8, and max_depth=4 were sufficiently constraining. ILP should inform the author when memory limits are being exceeded so the search parameters can be adjusted.

The longest run consumed about 7.5 hours of cpu time. This is quite acceptable, since it seems unlikely that ILP would encounter more than a few hundred examples in a day.

5.8 Guidelines for the use of ILP

The following guidelines for interacting with ILP were suggested by further experiments with the artificial cases:

- Nominal attributes with large value sets will swamp the search process and should be avoided.

---

[22] A practical implementation would allow the user to set the maximum number of examples saved. When this maximum is exceeded the oldest example would be deleted.

- The attributes should be restricted to the few judged by the author to be most relevant.
- Descriptions entered by the author before any examples are available are quite unstable. For this reason the initial stages of the learning process should be monitored closely, to enable the author to re-enter descriptions when deemed necessary.
- Failure of the learning process is generally indicated by poor completeness, consistency, and parsimony of the learned descriptions. When this situation cannot be remedied by entering new descriptions, the author is advised to develop a new set of attributes.

5.9 Future research

The instability of initial descriptions in the presence of noisy examples was the major shortcoming of ILP revealed by the experiments. One possible solution is to combine description initialization with Lee and Ray's (1986b) example set initialization method. Artificial examples generated from the description would lend stability until enough real examples were acquired, and thus prevent a description from being struck down prematurely by the first few invalid examples. To avoid the perpetuation of an incorrect initial hypothesis, the artificial examples could be gradually deleted as real examples are encountered.

Another shortcoming of ILP is that the clauses it produces tend to contain more terms than necessary. The bias toward comprehensibility should encompass a preference, not only for the least number of clauses, but also for clauses with the least number of terms. This problem is particularly acute when examples are non-uniformly distributed over the value set of an attribute. A second part could be added to the learning phase which would attempt to simplify CURDESC by deleting terms.

A further improvement could be the use of indirect examples in the learning process. Recall that indirect examples are those which can be inferred from the learning hierarchy. Suppose that one out of k prerequisites is recommended and proves to be non-mastered, resulting in a single positive example. After being instructed on the prerequisite, the student is returned to the prime objective which is passed without further difficulty. At this point k-1 indirect negative examples can be supplied to ILP. As has already been discussed, the validity of such indirect examples rests heavily on the validity of the learning hierarchy. Although the author may want to use indirect examples, he would regard them as somewhat less credible than direct examples. Thus, the author should be allowed to assign a weight to indirect examples so they could be included in the preference criterion. This weight would presumably reflect his degree of confidence in the learning hierarchy.

ILP is entirely based on the assumption that the factors underlying certain aspects of student behavior can be captured by relatively simple logical expressions composed of available information about the student. Only field testing with several real instructional systems and hundreds of students will bring sufficient evidence to bear on this premise.

5.10 References

Ashlock, R. (1976). *Error patterns in computation*. Columbus OH: Merril.

Hunka, S. (1987). *A survey of CAI capabilities of AVCs, colleges and technical institutes*. Division of Educational Research Services Research and Information Report (RIR-87-7). University of Alberta.

Quinlan, J. R. (1986). The effect of noise on concept learning. In R. Michalski, J. Carbonell, T. Mitchell (Eds.). *Machine learning: An artificial intelligence approach* (Vol. 2). Los Altos CA: Kaufmann.

Chapter 6

Application and Further Development of Memory Load Sequencing, ILP, and Related Procedures

The procedures developed in this thesis were designed to be supported by future systems facilitating the design, implementation and maintenance of courseware. In comparison with present authoring systems, such systems will be necessarily large and complex. They are likely to include:

- On-line expertise providing guidance in instructional design and use of the system.
- Specialized editor/compilers for various types of authoring activities.

Memory load sequencing and ILP are probably not practical for general use unless both these forms of support are available.

One can imagine that authors will define learning objectives within an objective editor, and will be advised on such questions as whether to unpack an objective into sub-objectives and whether to categorize an objective as cognitive, affective or psychomotor (as described by Jones and Massey-Hicks, 1987). Similarly, editors and on-line expertise would be available for aiding the author in defining the global structure of the course. Several different prerequisite relations would probably be available. For example, there may be a distinction between integral and non-integral prerequisites as discussed on page 51.

The editor which supports the definition of global course structure would have access to a syntax of admissible structure and would accept only syntactically correct relations. For example, in accordance with the theory of learning hierarchies, a verbal knowledge objective would not be permitted to have learning hierarchy prerequisites. As discussed in Chapter 3, such an editor would ensure that learning hierarchies entered by an author were acyclic, and would also be capable of assigning a level number to each prerequisite.

The procedures developed in this thesis are only a small portion of those that may be necessary to fully exploit learning hierarchies. Let us establish a simple sequence of loosely defined phases for organizing procedures of this type[23]:

1) Planning. Given some estimate of the student's current mastery of the learning objectives, procedures in this phase try to obtain a totally ordered set of instructional treatments or modules (i.e., a plan) which will enable the student to achieve the goals of the course in the shortest possible time. If the student is judged to have satisfied the goals of the course then exit, otherwise go to phase 2.
2) Treatment. The student enters the first module of the plan. If the student passes the module then go to phase 1 to get a new plan, otherwise go to phase 3.
3) Diagnosis. Using knowledge of the student and of global course structure, and mastery tests associated with each objective, procedures in this phase try to discover the student's mastery of objectives relevant to the plan failure. ILP is one approach to this problem. Go to phase 1 for a new plan.

6.1 Application and development of planning procedures

Even if only conjunctive learning hierarchies are permitted, and there is a one-to-one correspondence between modules and objectives, the memory load sequencing procedure described in Chapter 3 is not sufficient for the planning phase. Each iteration of the planning phase should be capable of creating a plan based on new information about the student (e.g., actual learning times) gathered in the treatment and diagnostic phases, but the current version of the memory load sequencing procedure cannot form a plan from a partially completed sequence. Overcoming this limitation is the next logical step in the development of memory load sequencing.

What additional modification to the planning phase can be imagined in order to accommodate the AND/OR hierarchies discussed on page 52? Assume that for each student and each objective/module there is an estimate of learning time, perhaps obtained by multiple regression. Since the goal is to minimize learning time, an algorithm could be developed which extracted, from the AND/OR hierarchy, the conjunctive hierarchy with the minimum estimated learning time. The extracted hierarchy could then be subjected to memory load sequencing to obtain a plan.

---

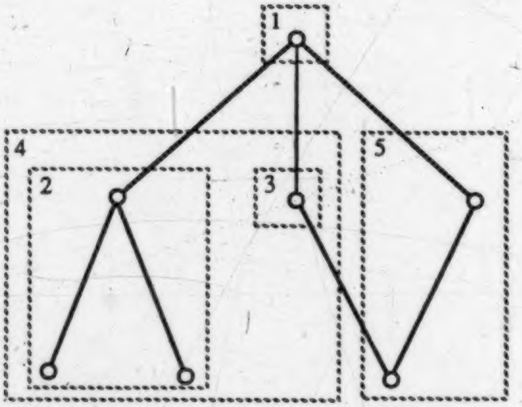
[23] This is really just a version of Hartley's framework (page 11).

A many-to-many relation between modules and objectives has been assumed in previous AIS such as Smallwood's teaching machine (page 15) and BIP (page 33). Theoretically, the advantages are that:

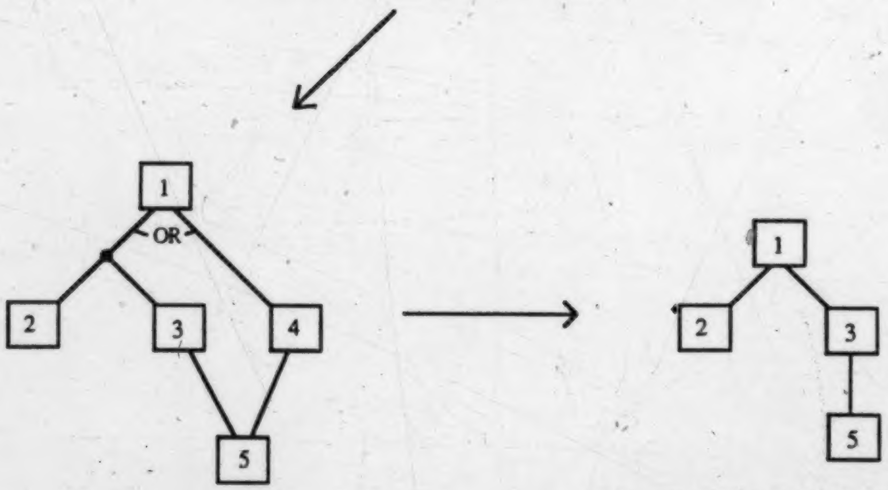
- The more capable students can be assigned to a module covering a set of objectives, some of which are not presented explicitly and are to be inferred from the objectives which are presented explicitly.
- The instructional treatment can be more natural because the author is not constrained to creating modules which teach single objectives in isolation.
- An objective can be taught in a variety of ways, and a student can be assigned to a module which best serves his or her learning style.

In practice, these advantages can only be realized when a repertoire of ATIs (attribute-treatment or achievement-treatment interactions) is established by ATI research. Mastery-based instructional systems seem better served by ATI research which adopts learning time rather than achievement as the outcome measure.

Figure 6.1a represents a structure that might be defined by an author within an editor allowing the many-to-many correspondence between modules and objectives. The author supplies for each objective a list of prerequisite objectives, and for each module a list of objectives. In Figure 6.1a, objectives are shown as circles and modules are shown as dashed boxes.



a. conjunctive learning hierarchy partitioned by modules



c. conjunctive module hierarchy

b. AND/OR module hierarchy

Figure 6.1 Transforming a conjunctive learning hierarchy into a conjunctive module hierarchy.

This kind of structure introduces further complications to the planning phase. One approach is illustrated in Figure 6.1. It assumes that learning times are associated with modules and that one starts with a conjunctive learning hierarchy. An intermediate representation, which might be called an AND/OR module hierarchy (Figure 6.1b), provides a more explicit representation of alternate plans. From it one extracts the conjunctive module hierarchy (Figure 6.1c) with the shortest estimated learning time. The memory load sequencing method described in Chapter 3 cannot be applied directly to the conjunctive module hierarchy because the assumption that objectives are not repeated may be violated. Thus a modified definition of memory load, likely requiring additional assumptions, may have to be adopted.

6.2 Application and development of diagnostic procedures

AND/OR hierarchies seem to present no additional difficulty to the use of ILP in the diagnostic phase. If the purpose of the diagnostic phase is regarded as only gathering information relevant to failure of the current plan, then ILP need consider only the most recently extracted conjunctive hierarchy (i.e., the one from which the plan was formed).

One can imagine ILP functioning in an environment where there is a many-to-many relation between objectives and modules, and where only module tests, not tests of individual objectives, are available. However, its use is best restricted to applications where every objective is associated with a distinct test or item pool.

Procedures for reasoning about prerequisite relations could be a useful addition to the diagnostic phase. For example, Heines and O'Shea (1985) suggested that partial evidence for the mastery or non-mastery of a prerequisite might be provided by records of performance on other objectives having the same prerequisite. One would expect the performance of this kind of procedure to deteriorate where there is more than one objective covered by a module.

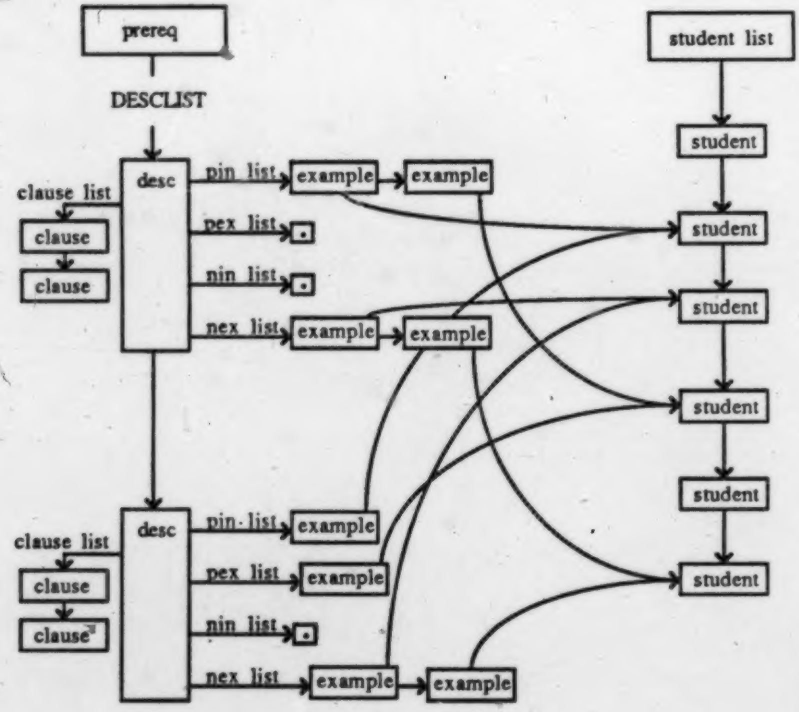
6.3 References

Heines, J., & O'Shea, T. (1985). The design of a rule-based tutorial. *International Journal of Man-Machine Studies*. 23, 1-25.

Jones, M., & Massey-Hicks, M. (1987). Expert CML: The next generation. paper presented at the Computer-Assisted Learning in Tertiary Education (CALITE) Conference, Sydney, Australia, November 30 - December 2, 1987.

Appendix A

This diagram is a simplified illustration of some of the data structures active during ILP's beam search. It shows the DESCLIST for one prerequisite after being truncated to a max_width of two. The DESCLIST contains two descriptions, each with its own example lists and clause list. There are six students shown in the student list, but only four were tested on the prerequisite. Therefore each description points to four examples. Each example points to one of the students in the student list. The student data structures contain the attribute vectors. Notice that the description at the top of the desclist is more complete than the second description in that it includes one positive example which is excluded by the second.


prereq
DESCLIST
student list
student
clause list
desc
pin list
example
example
student
clause
pex list
clause
nin list
student
nex list
example
example
student
student
desc
clause list
pin list
example
student
pex list
example
clause
clause
nin list
nex list
example
example

Appendix B

Case 1

number of students: 50

number of attributes: 6

attributes: $a_1, a_2, ..., a_6$ nominal attributes with value sets $\{0,1\}$

number of prerequisites: 3

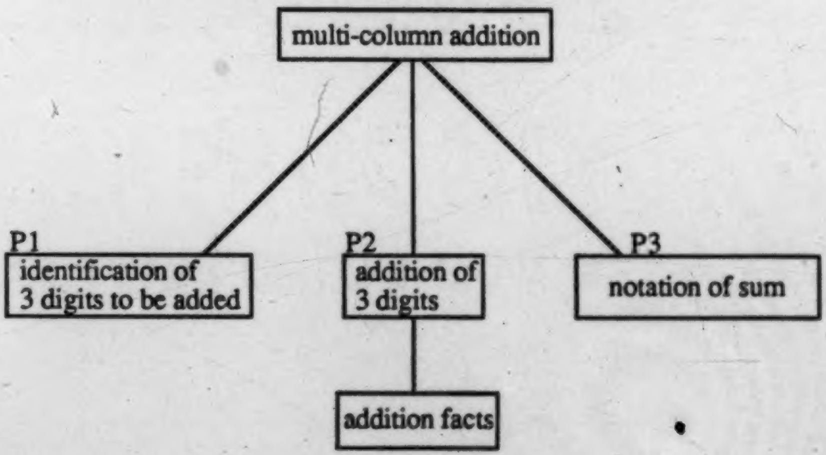
target description for P1: $[a_1=1 \ \& \ a_2=1 \ \& \ a_3=1] \vee [a_1=1 \ \& \ a_2=1 \ \& \ a_4=1]$

target description for P2: $[a_2=1 \ \& \ a_3=0 \ \& \ a_4=0] \vee [a_2=1 \ \& \ a_5=1 \ \& \ a_6=1]$

target description for P3: $[a_3=0 \ \& \ a_4=0 \ \& \ a_5=1] \vee [a_3=0 \ \& \ a_4=0 \ \& \ a_6=1]$

Case 2

course: Multiple column addition with two addends. The prime objective incorporates three prerequisites (P1, P2, P3) corresponding to the three sequential steps of the addition procedure that is taught. P1 teaches the student to identify three single digits in the current column (two digits from the addends and one digit from the carry row which is either 0 or 1). P2 teaches the addition of three digits in columnar format. P2 has the basic addition facts as its prerequisite, so any forgetting of these will cause failure of the P2 post-test. P3 shows how to write down the result of the three digit addition, including what to do with the carry digit.


multi-column addition
P1
identification of 3 digits to be added
P2
addition of 3 digits
P3
notation of sum
addition facts

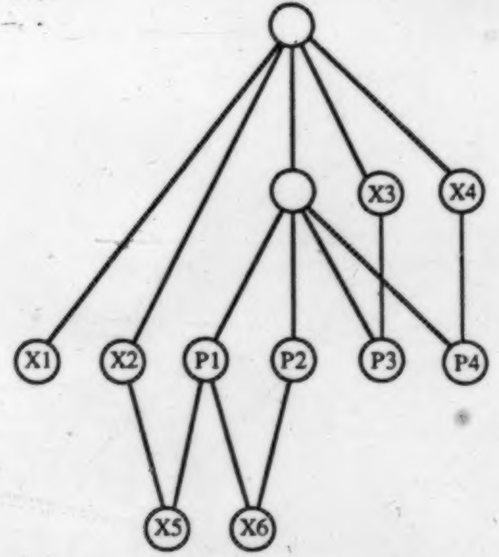
number of students: 100

number of attributes: 6

attributes:	In this case the attributes were simply the responses to six addition questions posed in the post-test of the goal objective. All six were treated as ordered attributes with value sets {0,1,...9999}.
error patterns:	40% percent of students were assumed to exhibit forgetting of two randomly selected addition facts. The false addition facts used were randomly generated with the caveat that the final false sum must be greater than the largest addend. In addition to the forgetting of addition facts, four common addition error patterns described in R. B. Ashlock's *Error Patterns in Computation* were used in generating the student models. They were left-to-right, digits-in-adjacent-col, borrows-right-bot, write-carry-in-sum. A description and example of each follows.
left-to-right:	The student starts adding the left column and moves right. 15% of students failing the goal post-test were assumed to exhibit this pattern. Example: 74 43 --- 18
digits-in-adjacent-col:	When there is no carry digit to be added (the carry digit is zero) the student takes a digit from the adjacent column on the left side. 15% of students failing the goal post-test were assumed to exhibit this pattern. Example: 46 3 --- 13
borrows-right-bot:	When there is no current column digit in the bottom addend (i.e., it is zero) then take the closest non-zero digit from the bottom addend. 15% of students failing the goal post-test were assumed to exhibit this pattern. Example: 75 8 ---- 163
write-carry-in-sum:	The student neglects to separate the carry portion of the column addition. 15% of students failing the goal post-test were assumed to exhibit this pattern. Example: 74 56 ------ 1210
target descriptions:	Addition fact errors would be detected by the P2 post-test, write-carry-in-sum errors would be detected by the P1 post-test, and the remaining three error patterns would be detected by the P3 post-test.

Case 3

course: Undetermined topic. This case is built on the structure of the following hierarchy (assume that X1 interferes with P2):



number of students: 400

number of prerequisites: 4

number of attributes: 12 ordered attributes

attributes:	value set	distribution
verbal aptitude (VAP)	{70...130}	normal with mean=100 sd=16
spatial aptitude (SAP)	{70...130}	normal with mean=100 sd=16
age in months (AGE)	{40...200}	normal with mean=120 sd=12
sex (SEX)	{male,female}	uniform
time since mastery of P1 (TMP1)	{1...100}	skewed with mode=5 df=2
time since mastery of P2 (TMP2)	{1...100}	skewed with mode=5 df=2
time since mastery of P3 (TMP3)	{1...100}	skewed with mode=5 df=2
time since mastery of P4 (TMP4)	{1...100}	skewed with mode=5 df=2
mastered X1 (X1)	{yes,no}	uniform
mastered X2 after P1 (MX2P1)	{yes,no}	uniform
mastered X3 (MX3)	{yes,no}	uniform
mastered X4 (MX4)	{yes,no}	uniform

target description for P1:

[TMP1=12..100 & TMP2=12..100] ∨ [TMP1=20..100 & MX4P1=no] ∨ [VAP=70..95 & TMP1=16..100]

P1 and P2 both have unit X6 as a prerequisite. Thus practice on P2 aids memory of P1. P1 and X2 both have X5 as a prerequisite. Thus practice on X2 aids memory of P1. Low verbal aptitude accelerates the forgetting of X5.

target description for P2:

[TMP1=12..100 & TMP2=12..100] v [TMP2=24..100] v [TMP1=20..100 & MX1=yes]

P1 and P2 both have unit X6 as a prerequisite. Thus practice on P1 aids memory of P2. Practice of X1 accelerates forgetting of P2.

target description for P3:

[SAP=70..72 & MX3=no] v [TMP3=20..100 & MX3=no]

Mastery of X3 consolidates memory for P3. Students with low spatial aptitude forget P3 immediately unless X3 is entered. Others can retain P3 for 20 hours.

target description for P4:

[VAP=70..95 & AGE=40..96 & TMP4=10..100] v [TMP4=20..100 & MX4=no] v

[TMP4=24..100]

Young students with low verbal aptitude forget P4 rapidly. Mastery of unit X4 aids memory for P4.

Case 4

course:	Advanced English grammar for students who are already fluent in English. Initial mastery of objectives is defined as a score of 80% or greater on the objective post-test. Each objective covers one grammatical rule which is presented in one of three modes (definition,example,conversation). Definition mode is an expository frame defining the rule. Example mode is a frame presenting a few isolated examples of the rule. Conversation mode is an example embedded in a conversation on some topic initiated by the program (e.g., sports), in which the student is expected to repeat the usage of the rule in his response. All modes are followed by practice of the rule in the post-test. The target population includes some students who have English as a first language, some having English as a second language, and some who have two first languages (bilingual), one of which is English. Also, each student is rated as reflective, normal, or impulsive according to performance on a test of impulsivity/reflectivity.
number of prerequisites:	3
number of attributes:	15

attributes:	range	distribution

Time since mastery of P1 (TMP1)	0...100 hours	skewed with mode=20 df=4
Time since mastery of P2 (TMP2)	0...100 hours	skewed with mode=20 df=4
Time since mastery of P3 (TMP3)	0...100 hours	skewed with mode=20 df=4
Learning time of P1 (LTP1)	1...100 minutes	skewed with mode=10 df=2
Learning time of P2 (LTP2)	1...100 minutes	skewed with mode=10 df=2
Learning time of P3 (LTP3)	1...100 minutes	skewed with mode=10 df=2
Degree of mastery of P1 (DMP1)	80...100 %	skewed with mode=85 df=3
Degree of mastery of P2 (DMP2)	80...100 %	skewed with mode=85 df=3
Degree of mastery of P3 (DMP3)	80...100 %	skewed with mode=85 df=3
Presentation Mode of P1 (MOP1)	{def,exam,con}	nominal
Presentation Mode of P2 (MOP2)	{def,exam,con}	nominal
Presentation Mode of P3 (MOP3)	{def,exam,con}	nominal
English Background (ENG)	{first,sec,bil}	nominal
Impulsive/Reflective (IR)	{ref,norm,imp}	nominal
Sex (SEX)	{male,female}	nominal

target description for P1:

[TMP1=65..100 & DMP1=80..90] ∨

[TMP1=40..100 & LTP1=1..6 & DMP1=80..85] ∨
[TMP1=30..100 & LTP1=1..4 & DMP1=80..83]

Students forget P1 if the time since mastery is greater than or equal to 65 hours and the degree of mastery was less than 90%. They will also forget P1 when it has been mastered more recently if they spent little time studying it and if their degree of mastery is minimal.

target description for P2:

[LTP2=1..3 & MOP2=def & ENG=first] ∨ [LTP2=1..3 & MOP2=con & ENG=sec]

Students forget P2 if the time spent studying it is less than 3 minutes and one of the following conditions is met. Students having English as their sole first language are unfamiliar with the application of formal grammar, so will forget P2 when it is presented under the definition mode. Students who learned English as second language have trouble generalizing the single example presented in conversation mode.

target description for P3:

[LTP3=1..7 & MOP3=con & IR=ref & SEX=female] ∨ [LTP3=1..7 & MOP3=def & IR=imp & SEX=male]

Students forget P3 if the time spent studying it is less than 7 minutes and one of the following conditions is met. The conversation mode for this objective involves sports, a subject which is not memorable for reflective females. When learned by definition mode, this objective is easily forgotten by impulsive males.